21世纪高等学校计算机基础实用规划教材

算法与程序设计基础

（Python版）

吴　萍　主编
朱晴婷　蒲　鹏　副主编
刁庆霖　裘奋华　等　编著

清华大学出版社

21世纪高等学校计算机基础实用规划教材

算法与程序设计基础
（Python版）

吴　萍　主编

朱晴婷　蒲鹏　副主编

刁庆霖　裘奋华　等编著

清华大学出版社

北　京

内 容 简 介

理论、思维训练与实践相结合是本书的特色。本书共分为8章，通过将算法与程序设计的基本概念与某种程序设计语言(以Python为主)结合使学生理解计算思维的概念，了解算法与程序的关系，能够进行较为简单而经典的算法设计、评价算法的性能与效率，并能利用程序设计语言进行简单的程序开发；培养学生利用计算机解决与专业、科研、社会需要密切相关的实际问题的能力和基本创新精神，以适应信息化社会的要求、拓宽专业生存空间，使其在后续专业课程的学习和未来的工作中较长期地受益。

本书是高等院校非计算机专业的计算机技术应用课程的教材，适用于文理工科各专业学生，也可作为初学者的入门书籍。

图书在版编目(CIP)数据

算法与程序设计基础：Python版/吴萍主编.--北京：清华大学出版社，2015

21世纪高等学校计算机基础实用规划教材

ISBN 978-7-302-39021-3

Ⅰ.①算…　Ⅱ.①吴…　Ⅲ.①电子计算机－算法理论－高等学校－教材 ②程序设计－高等学校－教材 ③软件工具－程序设计－高等学校－教材　Ⅳ.①TP301.6 ②TP311.1 ③TP311.56

中国版本图书馆CIP数据核字(2015)第017205号

责任编辑： 付弘宇　薛　阳

封面设计：

责任校对： 李建庄

责任印制：

出版发行： 清华大学出版社

网　　址： http://www.tup.com.cn，http://www.wqbook.com

地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084

社 总 机： 010-62770175　　**邮　　购：** 010-62786544

投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn

质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn

课件下载： http://www.tup.com.cn，010-62775954

印 刷 者：

装 订 者：

经　　销： 全国新华书店

开　　本： 185mm×260mm　　**印　张：** 18.5　　**字　　数：** 460千字

版　　次： 2015年2月第1版　　**印　　次：** 2015年2月第1次印刷

印　　数： 1～　000

定　　价： .00元

产品编号：061819-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程（简称'质量工程'）"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程（群）加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机公共课程领域，以公共基础课为主、专业基础课为辅，横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

（1）面向多层次、多学科专业，强调计算机在各专业中的应用。教材内容坚持基本理论适度，反映各层次对基本理论和原理的需求，同时加强实践和应用环节。

（2）反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生的知识、能力、素质协调发展创造条件。

（3）实施精品战略，突出重点，保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现教学质量和教学改革成果的教材。

（4）主张一纲多本，合理配套。基础课和专业基础课教材配套，同一门课程可以有针对不同层次、面向不同专业的多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

（5）依靠专家，择优选用。在制定教材规划时依靠各课程专家在调查研究本课程教材

建设现状的基础上提出规划选题。在落实主编人选时,要引入竞争机制,通过申报、评审确定主题。书稿完成后要认真实行审稿程序,确保出书质量。

繁荣教材出版事业,提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度,希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21世纪高等学校计算机基础实用规划教材

联系人:魏江江 weijj@tup.tsinghua.edu.cn

前　言

为了适应信息和计算技术的发展，切实满足社会各个领域对计算机应用人才不断增长的需求。本书设计了“算法与程序设计基础”的通识课程方案，力求融入计算思维的思想，将多年来计算机学科所形成的解决问题的思维模式和方法渗透到各个学科。与传统的程序设计类教材不同，本书选择较容易上手的 Python 语言，着重介绍分析问题和解决问题的方法和思路，通过对不同解决方案的分析比较，让学生掌握选取优化方案并予以实现的理论方法和实际应用能力。

本教材编写具有以下特点。

1. 难点和重点安排合理

教材中的内容编排凝聚了作者多年的教学经验与体会，并在章节的篇幅和安排上为教师提供了讲解内容和时间安排上的灵活性。扩展部分使有能力的读者可以更上一层楼，并把本书当作一个有价值的参考资源。

2. 讲解深刻

对一些重难点知识，学生不仅要知其然，还需要知其所以然。因此教材中会为教师和学生剖析其本质，让学生能够从根本上理解、掌握并灵活运用这些知识。

3. 实用性强

教材中提供了大量针对性的实例，同时编程中要注意什么？如何阅读出错提示？出现问题如何解决？教材中都会一一讲解，带领学生迅速掌握编程的全过程。

4. 涵盖了算法和程序设计的较为核心的内容

选择了经典和应用广泛的各类算法，并结合程序设计的思想和方法，让学生能够通过循序渐进的程序设计过程了解计算的魔力，掌握求解问题的方法，进而融入后续的学习和今后的生活和工作中。

本书由吴萍负责全书的统稿。第 1 章由朱敏、陈志云、蒲鹏执笔，第 2 章、第 6 章由周力执笔，第 3 章由朱晴婷执笔，第 4 章由蒲鹏执笔，第 5 章由朱晴婷、裘奋华执笔，第 7 章由吴萍执笔，第 8 章由刁庆霖执笔。附录 A 由各章编写者提供，附录 B 由郑凯选编。

由于时间仓促和作者水平有限，书中难免有不妥之处，恳请广大读者批评指正。

本书的配套课件等资源可以从清华大学出版社网站 www.tup.com.cn 下载，下载或使用中的相关问题，请联系 fuhy@tup.tsinghua.edu.cn。

编　者

2015 年 1 月

于华东师范大学

目　录

第1章 程序设计与计算思维

计算机虽然是人类制造出来的，但是它还是依托电子器件而存在的，离不开它本身固有的电子属性；所以在使用程序设计指导计算机工作的时候，就必须建立一种特殊的能适合计算机思考的思维方法。

学习程序设计语言，不仅要掌握其语法规则，更重要的是从思维训练入手，学会分析问题，从实际问题中抽象求解方法的能力。

1.1 程序设计与计算机语言

1.1.1 程序设计

程序设计(Programming)是给出解决特定问题程序的过程，是设计、编制、调试程序的方法和过程。它是目标明确的智力活动，是软件构造活动中的重要组成部分。它以某种程序设计语言为工具，给出这种语言下的程序。程序设计通常分为问题分析、算法设计、程序编写、程序运行、结果分析和文档编写等阶段。专业的程序设计人员常被称为程序员。

1.1.2 设计步骤

(1) 问题分析。

对于接受的任务进行认真的分析，研究给定的条件，分析最后应达到的目标，找出解决问题的规律，选择解决问题的方法，完成实际问题。

(2) 算法设计。

设计出解决问题的方法和具体步骤。

(3) 程序编写。

根据设计的算法，选择一种程序设计高级语言编写出源程序，并通过测试。

(4) 对源程序进行编辑、编译和连接。

(5) 运行程序，分析结果。

运行可执行程序，得到运行结果，并对结果进行分析，看它是否符合要求。如不符合要求，就需要进行修改、再测试、再运行，直至结果正确。

(6) 文档编写。

文档编写内容应包括程序名称、程序功能、运行环境、程序的装入和启动、程序的输入、输出数据，以及使用注意事项等。

1.1.3 程序设计分类

按照结构性质分类,程序设计有结构化程序设计与非结构化程序设计之分。结构化程序设计是指具有结构性的程序设计方法与过程,它具有由基本结构构成复杂结构的层次性,非结构化程序设计反之。

按照用户的要求分类,程序设计有过程式程序设计与非过程式程序设计之分。过程式程序设计是指使用过程式程序设计语言的程序设计,非过程式程序设计指非过程式程序设计语言的程序设计。

按照程序设计的成分性质分类,程序设计有顺序程序设计、并发程序设计、并行程序设计、分布式程序设计之分。

按照程序设计风格分类,程序设计有逻辑式程序设计、函数式程序设计、对象式程序设计之分。

1.1.4 基本规范

程序设计规范是进行程序设计的具体规定。程序设计是软件开发工作的重要部分,而软件开发是工程性工作,所以必须有规范,才能保证程序设计的质量。

1.1.5 计算机语言

语言分为自然语言与人工语言两大类。自然语言是人类在自身发展的过程中形成的语言,是人与人之间传递信息的媒介。人工语言指的是人们为了某种目的而自行设计的语言。计算机语言就是人工语言的一种。计算机语言是人与计算机之间传递信息的媒介。为了使电子计算机能进行各种工作,就需要有一套用以编写计算机程序的数字、字符和语法规划,由这些数字、字符和语法规则组成计算机的各种指令(或各种语句)就是计算机能接受的语言。

1.1.6 语言分类

计算机语言的种类很多,可以分成机器语言、汇编语言、高级语言三大类。

机器语言是表示成数码形式的机器基本指令集,或者是操作码经过符号化的基本指令集。汇编语言是机器语言中地址部分符号化的结果,或进一步包括宏构造。高级语言的表示方法更接近于待解问题的表示方法,其特点是在一定程度上与具体机器无关,易学、易用、易维护。

程序设计语言按照用户的要求可分为过程式语言和非过程式语言。过程式语言的主要特征是,用户可以指明一列可顺序执行的运算,以表示相应的计算过程,如 FORTRAN、COBOL、PASCAL 等。

按照应用范围可分为通用语言与专用语言,如 FORTRAN、COLBAL、PASCAL、C 语言等都是通用语言。目标单一的语言称为专用语言,如 APT 等。

按照使用方式可分为交互式语言和非交互式语言。具有反映人机交互作用的语言称为交互式语言,如 BASIC 等。不反映人机交互作用的语言称为非交互式语言,如 FORTRAN、COBOL、ALGOL69、PASCAL、C 语言等都是非交互式语言。

按照成分性质可分为顺序语言、并发语言和分布语言。只含顺序成分的语言称为顺序语言，如 FORTRAN、C 语言等。含有并发成分的语言称为并发语言，如 PASCAL、Modula 和 Ada 等。

程序设计语言还可分为面向对象语言和面向过程语言，面向对象的例如：C++/C#/Java 等，面向过程的例如：Free Pascal/C 语言等。

1.2 计算机语言与计算思维的关系

思维和语言是紧密相连的社会现象，思维与语言之间究竟是什么关系，历来是语言学家、哲学家、心理学家、人类学家、生物学家、逻辑学家都十分关心的问题。

1.2.1 思维与计算思维

思维是什么？哲学上的思维通常指人脑对客观事物间接和概括的反映，是认识的理性阶段。中国、俄罗斯和东欧各国学术界普遍认为，思维是人借助语言实现的对客观事物概括、间接的反映，是反映对象本质和规律的认识过程。

计算思维(Computational Thinking)是运用计算机科学的基础概念进行问题求解、系统设计，以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。这个概念是由美国卡内基·梅隆(Carnegie Mellon)大学计算机科学教授周以真教授提出的，要理解它，首先得搞清楚什么是问题求解、系统设计和理解人类行为。

1. 问题求解

在每天的生活中，都会碰到很多问题，面对不同的问题，需要不断地进行思考和选择。而解决问题的方法也是多种多样的，不同的方法或许都能达到相同的目的，但过程和效率却可能大相径庭，最终如何解决问题，取决于对问题的分析和对所选方案的周密考虑。

【讨论】 开学了，作为学生会主席，你在考虑举行一场迎新晚会来迎接新同学的到来，那么，晚会该如何组织？如果申请的费用有限，如何在预算范围内，将晚会办得更好些？

这时，你也许会考虑这样一些问题：晚会的规模是多大？这取决于预算情况。会有多少老师和学生来参加？需要租借的场地也取决于此。还需要哪些资源？这些资源如何获取？把学生会干部们找来一起商量，根据大家的多个主意，权衡利弊，找到最佳解决方案。

问题求解的一般过程可以归纳为图 1-2-1。

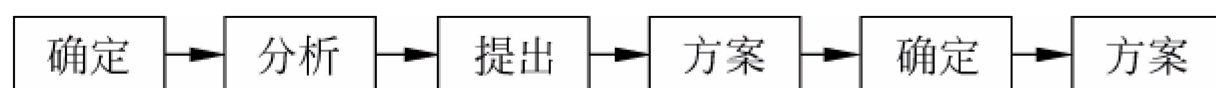


图 1-2-1 问题求解的一般过程

确定问题：这是一个很重要的环节，如果把问题理解错了，后面的工作都是无效的。例如：要解决预算范围内的晚会规模，就不能只考虑将晚会办得华丽了。

分析问题：在这一环节，可以通过了解问题的相关背景知识，来进一步理解问题。如一

般开一个晚会碰到的问题有哪些？场地、人员、节目、服装等，目前已有了哪些资源，还有哪些方面需要解决。预算费用如何在不同的需要方面进行划分，如何在有限的费用前提下将晚会办得尽可能规模大些，档次高些。

通过问题的分析提出方案，在这个过程中，有可能会有多种方案产生。可以在各种方案的基础上权衡利弊，选择最适合的方案。

方案一旦确定后，可以写出该方案的实施步骤，步骤由简单到详细，可以逐步细化，直到容易实施。

在方案实施以后，可以对方案的效果做出评价，以便对今后类似问题的解决有所借鉴。

如果要用计算机来解决问题，同样需要以上过程进行问题求解。利用计算手段求解问题的过程是：首先要把实际的应用问题转换为数学问题，然后建立模型、设计算法和编程实现，最后在实际的计算机中运行并求解。前两步是抽象的过程，后两步则是自动化的过程，而计算思维的本质便是抽象与自动化，这种思维方式是在计算科学的基础上逐渐建立起来的，通过计算科学领域的学习，可以使我们的计算思维得到提升，并扩展到学习、生活的其他方面。

2. 系统设计

任何自然系统和社会系统都可视为一个动态演化系统，演化伴随着物质、能量和信息的交换，这种变换可以映射为符号变换，使之能用计算机实现离散的符号处理。当动态演化系统抽象为离散符号系统后，就可以采用形式化的规范来描述，通过建立模型、设计算法和开发软件来揭示演化的规律，实时控制系统的演化并自动执行。

例如：计算机网络系统是一个复杂的系统，为了实现该系统，人们通过分层设计的方法，将一个复杂的系统变成多个层次，每个层次都有清晰的功能与上下层次之间的接口，这样，人们的精力可以放在对每个层次的设计上，使复杂的问题变得简单。这种分层设计的方法经常会用在复杂系统的设计上，通过不断分层，直到将复杂问题简化为可以通过某种数学方法和模型将问题得到解决。这种思维方式不仅可以应用在计算科学领域，也可推广到其他任何复杂系统的设计领域。

3. 人类行为

计算思维是基于可计算的手段，以定量化的方式进行的思维过程。在人类的物理世界、精神世界和人工世界这三个世界中，计算思维是建设人工世界所需要的主要思维方式。

利用计算手段来研究人类的行为，可视为社会计算(Cyber-Society Computing)，即通过各种信息技术手段，设计、实施和评估人与环境之间的交互。社会计算涉及人们的交互方式、社会群体的形态及其演化规律等问题。研究生命的起源与繁衍、理解人类的认知能力、了解人类与环境的交互以及国家的福利与安全等，都属于社会计算的范畴。这些都与计算思维密切相关。

【讨论】 互联网的发展使用户数量激增，大量用户的网络使用有什么规律可循吗？怎样利用网络上的信息动态来把握商机？通过讨论，理解计算思维与人类行为的关系。

1.2.2 计算思维与计算科学的关系

计算思维虽然源自计算科学的发展，拥有计算科学领域的许多特征，看似与计算机相

关，但是计算思维本身并不是计算科学的专属，更不能认为只有与计算机相关的，才具有计算思维。实际上，即使没有计算机，计算思维也会逐步发展，甚至有些内容与计算机没有关联。但是，正是由于计算机的出现，给计算思维的研究和发展带来了根本性的变化。

由于计算机对信息和符号具有快速处理能力，使得许多原本只是理论上可以实现的过程变成了实际可以实现的过程。海量数据的处理、复杂系统的模拟和大型工程的组织，都可以借助计算机实现从想法到产品整个过程的自动化、精确化和可控化，大大拓展了人类认识世界和解决问题的能力和范围。机器替代人类的部分智力活动激发了人们对于智力活动机械化的研究热潮，凸显了计算思维的重要性，推进了计算思维的形式、内容和表述的深入探索。在这样的背景下，作为人类思维活动中以形式化、程序化和机械化为特征的计算思维受到人们的重视，并且本身作为研究对象也被广泛和深入地研究着。

随着计算科学和技术的发展，社会进入了大数据时代，人们可以轻易地获得大量数据，并有能力在短时间内对完整的数据进行分析，从而获得新的信息。谁能利用好大数据，谁就获得了先机，计算思维的培养推动了大数据时代的创新。

什么是计算，什么是可计算，什么是并行计算，计算思维的这些性质得到了前所未有的彻底研究。由此不仅推进了计算机的发展，也推进了计算思维本身的发展。在这个过程中，一些属于计算思维的特点被逐步揭示出来，计算思维与理论思维、实验思维的差别越来越清晰化。计算思维的内容得到不断的丰富和发展，例如在对指令和数据的研究中，层次性、迭代表述、循环表述以及各种组织结构被明确提出来，这些研究成果也使计算思维的具体形式和表达方式更加清晰。从思维的角度看，计算科学主要研究计算思维的概念、方法和内容，并发展成为解决问题的一种思维方式，极大地推动了计算思维的发展。

1.2.3 计算思维与程序设计语言的关系

计算科学不是计算机编程。像计算机科学家那样去思维意味着远远不仅限于计算机编程，还要求能够在抽象的多个层面上思维。因此概念化的抽象思维不只是程序设计。

计算思维是人类求解问题的一条途径，但绝非要使人类像计算机那样去思考。计算机枯燥且沉闷，人类聪颖且富有想象力。是人类赋予了计算机激情，配置了计算机设备，人们就能用自己的智慧去解决那些计算机时代之前不敢尝试的问题，达到“只有想不到，没有做不到”的境界。计算机赋予人类强大的计算能力，人类应该更好地利用这种力量去解决各种需要大量计算的问题。

计算科学在本质上源自数学思维，因为像所有科学一样，其形式化基础是建筑在数学之上的。计算科学又从本质上源自工程思维，因为人们建造的是能够与实际世界互动的系统，基本计算机设备的限制迫使计算机科学家必须计算性地思考，而不是只是数学性地思考。构建虚拟世界的自由使人们能够超越物理世界的各种系统。数学和工程思维的互补与融合很好地体现在抽象、理论和设计三个学科形态上。

而程序设计语言(Programming Language)是用于书写计算机程序的语言，包含一组记号和一组规则，根据规则由记号构成的记号串的总体就是语言。程序设计语言有三个方面的因素，即语法、语义和语用。语法表示程序的结构或形式，亦即表示构成语言的各个记号之间的组合规律，但不涉及这些记号的特定含义，也不涉及使用者。语义表示程序的含义，亦即表示按照各种方法所表示的各个记号的特定含义，但不涉及使用者。语用表示程序与

使用者的关系。

学习了程序设计语言,掌握了其语法规则,但并不一定就能编写好的程序,这就像一个孩子,会用母语说话,但并不会用它来写小说。

实际上,程序设计语言是为人们解决问题服务的工具。真正要让计算机能工作,能为人们服务,要有好的软件,而软件的设计与制作虽然离不开程序设计语言,但更主要的是人们所提出的好的系统设计、好的问题解决方案,对方案的合理细化,并通过某种合适的程序设计语言的编程实现。

学习程序设计,虽然需要掌握程序设计语言的语法规则,但更重要的,是从训练思维入手,学会问题分析、思考解决方案,学会对各种现实问题的抽象,探寻计算机自动化的实现规律,这样才能在掌握程序设计语言语法的基础上,编制出好的程序,让计算机实现有效的功能,为人们提供服务。我们一定要把握好学习程序设计课程的机会,有意识地培养和提升自己的计算思维能力,为自己在专业领域的创新打下基础。

1.3 初识 Python 语言

1.3.1 Python 语言概述

1. 起源

为了让大家了解 Python 的起源,这里摘录一段 Leo Laporte(Twitter 网站创始人)和 Guido Van Rossum 的访谈录。Guido Van Rossum 从 2005 年开始在职于 Google。这个长相酷似埃尔顿·约翰逊的 Guido Van Rossum(图 1-3-1)虽然没有谱写出《风中之烛》那样的名曲,却在 1991 年的时候,给开源社区贡献了性能卓著的编程语言 Python。他也因此荣膺为 Python 之父。这个单词原意为蟒蛇,中文发音一般读作“派桑”。



图 1-3-1　Python 作者

Leo Laporte:

Google 是 Python 的一个大用户。首先,要问一个问题,是关于 Python 起源的,如果我说错了,请大家更正一下。据我所知,最初 Python 的设计是为了教学的目的,想创建一个用于学习和编程的语言,这么说对吗?

Guido Van Rossum:

Python 从一个叫 ABC 的语言继承了很多东西,而 ABC 这种语言在设计的时候就特别考虑到用于教学。在 20 世纪 80 年代末期,1989 年的时候,我觉得自己有必要创建一门新语言,我借鉴了 ABC 语言中我所喜欢的特点,并将其中我不喜欢的东西用自己创新的或一些借鉴自别处的想法取而代之。我的目标是要建立一个为专业程序员使用的脚本语言,而这些专业程序员主要使用 C 语言和 Borne Shell 脚本语言作为他们的主要开发语言。Python 的位置大概是介于 C 和 Shell 语言之间的,所以我创建 Python 并没有明确的教学目的。因为我从 ABC 语言中借鉴了那么多,而 ABC 本身又有教学的目的在其中,所以我建立的语言也就很适合做教学语言。

Leo Laporte：

人们问我很多次，每次我都向他们推荐 Python，因为它是免费的，而且是跨平台的。使用它的解释器来教学编程很简单，你可以立刻尝试用解释器来学习语言。

Guido Van Rossum：没错，这是我从 ABC 借鉴来的，ABC 也具有这些特点。

Chris DiBona：那么 ABC 是不是也有严格的空白格式要求呢？

Guido Van Rossum：ABC 也有强制缩进的要求。

Chris DiBona：我发现一件事情，无论从项目、公司角度来看，Python 都更容易维护，从一个人到另外一个人，你同意这点吗？

Guido Van Rossum：是啊，我完全同意。这其实不是我专门设计语言的目的，很难说我这么设计的目的。

Leo Laporte：你用 C 写的 Python 吧？

Guido Van Rossum：是的，Python 的主要实现是用 C。

Guido Van Rossum：Python 在从其他语言上借鉴好想法这一点上是相当开放的，我从来也不羞于承认 Python 的哪些特点是从哪里来的，Python 的最大缺点大概就是它是由我发明的。

读者在耐心仔细读完这段访谈录之后，将会对 Python 有个直观印象，它师从 C 语言，但比前者更开放，更简洁。

在这段访谈之后没多久，Google 终究没有困住这位天才，他在自己的推特中宣布他将离开 Google，加盟 Dropbox（一个基于 Python 的云存储创业公司），如图 1-3-2 所示。

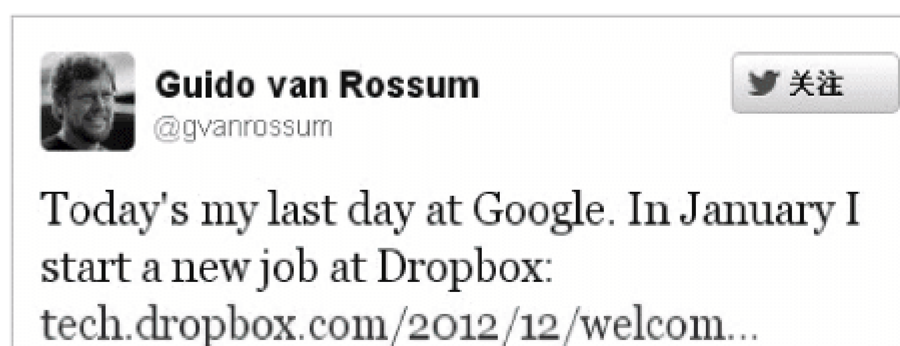


图 1-3-2　推特上的辞职宣告

2. 特点

从上面 Guido Van Rossum 的大段对话中，可以总结出 Python 具备以下的特点：

(1) 适合教学的脚本语言。

(2) 跨平台和兼容性非常好，可运行在多种计算机平台和操作系统中，如 UNIX，Windows，Mac OS，OS/2 等。

除此之外，它还具备以下的特点：

自动内存回收。这个特点使得程序员在编程的时候，可以不考虑程序运行中的内存管理，而专注于自己的逻辑处理。

面向对象特性（Object-Oriented）。这个特点使得 Python 语言顺应了当今程序设计语言发展的大势，从而为它被更加广泛地应用奠定了基础。它博采众长，支持多重继承（Multiple Inheritance），重载（override）。这些细节将在本书的后续章节中详细讲述。

强大的动态数据类型支持，不同数据类型相加会引发一个异常。

强大的类库支持，使编写文件处理、正则表达式，网络连接等程序变得相当容易。

Python 的交互命令行模块能方便地进行小代码调试和学习。

Python 易于扩展，可以通过 C 或 C++编写的模块进行功能扩展。

它是开源、免费的。一直长期被某些大公司垄断的 IT 领域，对于任何高喊开源、免费的技术或者产品，都会拥有超高的人气。开源、免费也被更多地看作一种崇尚自由的气质。

正因为它师从 C 语言,所以在运行效率上,它不如 Java 或者 C 代码高。

3. 为什么要学习 Python

可能有很多学习 Python 的理由摆在面前:对自由的崇尚,对编程之美的欣赏,简单至上的诱惑,等等。但对于非计算机专业的读者来说,它更多地是一个快速学习逻辑、掌握计算思维的工具。

“所有人在小学二年级已经学会了写作,然而大多数人必须从事其他更重要的工作。”——鲍比·奈特

这句话在本书中就可以理解为,虽然有些人对于编程很感兴趣,但是对于大多数人来说,编程仅是完成其他任务的工具而已。由于 Python 语言比较简单,让人们可以有更多时间理解它在解决问题中所体现的思路,以及处理数据的内在含义,而无需花费太多精力解决计算机如何得到数据结果,也就是说,它让我们更容易实现自己的目的。

1.3.2 Python 语言的应用

Python 从诞生之初,就受到了很多领域的青睐,发展至今,它已在下面这些领域得到了不同程度的应用。

- 系统编程,提供大量系统接口 API,能方便地进行系统维护和管理。
- 图形处理,有 PIL、Tkinter 等图形库支持,能方便地进行图形处理,开发 GUI(图形用户界面)应用程序。目前而言,使用 Python 开发 GUI 程序,除了 Tkinter(本书后续章节会介绍)之外,还可以使用 wxPython 图形库和大名鼎鼎 QT。wxPython 是 Python 语言的一套优秀的 GUI 图形库,允许 Python 程序员很方便地创建完整的、功能健全的 GUI 用户界面。它是以 Python 模块的方式提供给用户的,是一款开源软件,并且具有非常优秀的跨平台能力,能够支持运行在 32 位 Windows、绝大多数的 UNIX 或类 UNIX 系统、Macintosh OS X 下。在 Python 中输入下述的代码,运行便可弹出如图 1-3-3 所示的界面。

```
#! /usr/bin/env python
# - * - encoding: utf - 8 - * -
#author pythontab.com

#引入 wx 模块
importwx

#定义一个 wx 的 class
classsayHello(wx.App):

defOnInit(self):
frame = wx.Frame(parent = None, title = "Hello wxPython")
frame.Show()
return True
app = sayHello()
#主循环
app.MainLoop()
```

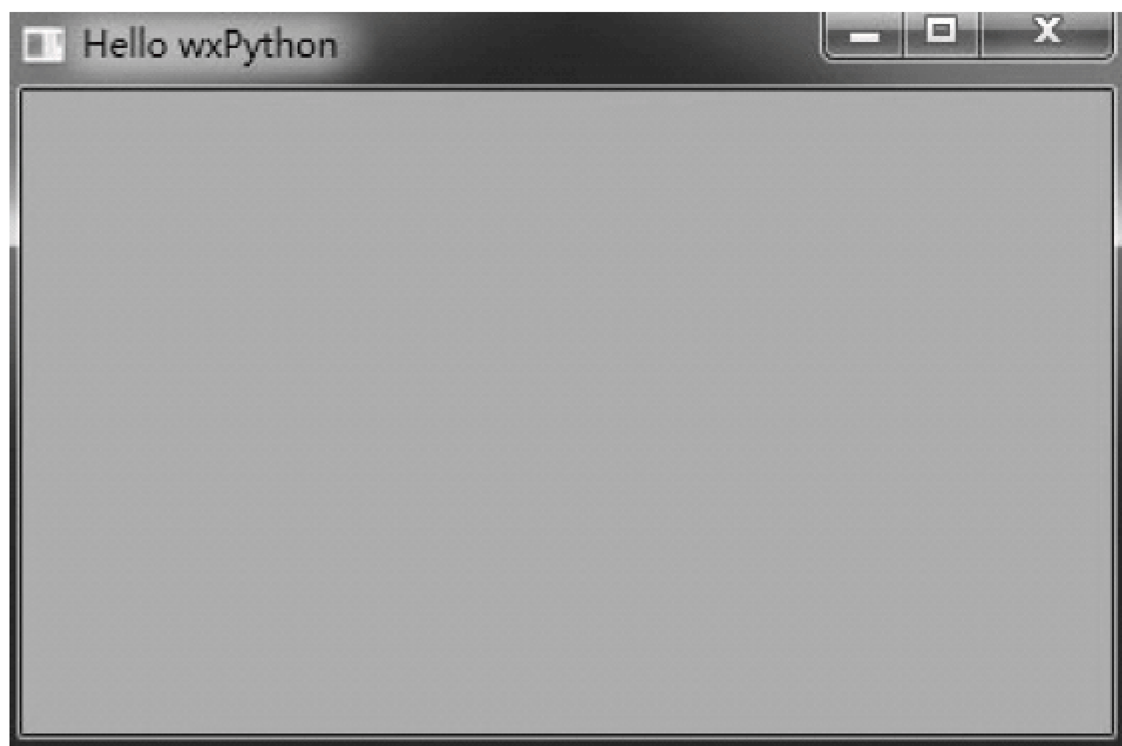


图 1-3-3　基于 wxPython 开发的 GUI 界面

- 数学处理，NumPy 扩展提供大量与许多标准数学库的接口。
- 文本处理，Python 提供的 re 模块能支持正则表达式，还提供 SGML，XML 分析模块，许多程序员利用 Python 进行 XML 程序的开发。
- 数据库编程，程序员可通过遵循 Python DB-API（数据库应用程序编程接口）规范的模块与 Microsoft SQL Server、Oracle、Sybase、DB2、MySQL 等数据库通信。Python 自带有一个 Gadfly 模块，提供了一个完整的 SQL 环境。
- 网络编程，提供丰富的模块支持 Sockets 编程，能方便快速地开发分布式应用程序。
- 作为 Web 应用的开发语言，支持最新的 XML 技术。
- 多媒体应用，Python 的 PyOpenGL 模块封装了 OpenGL 应用程序编程接口，能进行二维和三维图像处理。
- 近年来随着游戏产业的兴起，Python 开始越来越多地涉足游戏领域。Pygame 是 Python 开发游戏的一个库，关于 Pygame，具体可参考 http://www.pygame.org 网站。如图 1-3-4 所示，便是两个基于 Pygame 开发的游戏界面。

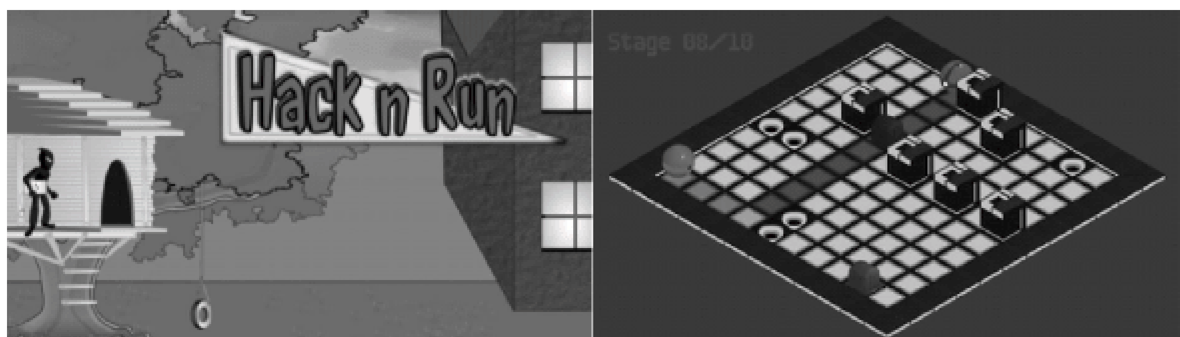


图 1-3-4　基于 Pygame 的游戏界面

1.3.3　编辑与运行环境

1. 下载和安装 Python

首先要下载 Python 语言解释器，本书下载的是 Python 3.3.0 的 Windows 安装版本，官方的下载地址是 http://www.Python.org/getit/，下载成功后，双击安装包，弹出如图 1-3-5 所示的安装画面。

在此画面中可以采用 Python 的默认路径，也可以自己改变，本安装示例就将 Python 的安装目录修改为 C:\Python33，然后单击 Next 按钮，弹出如图 1-3-6 所示的画面。

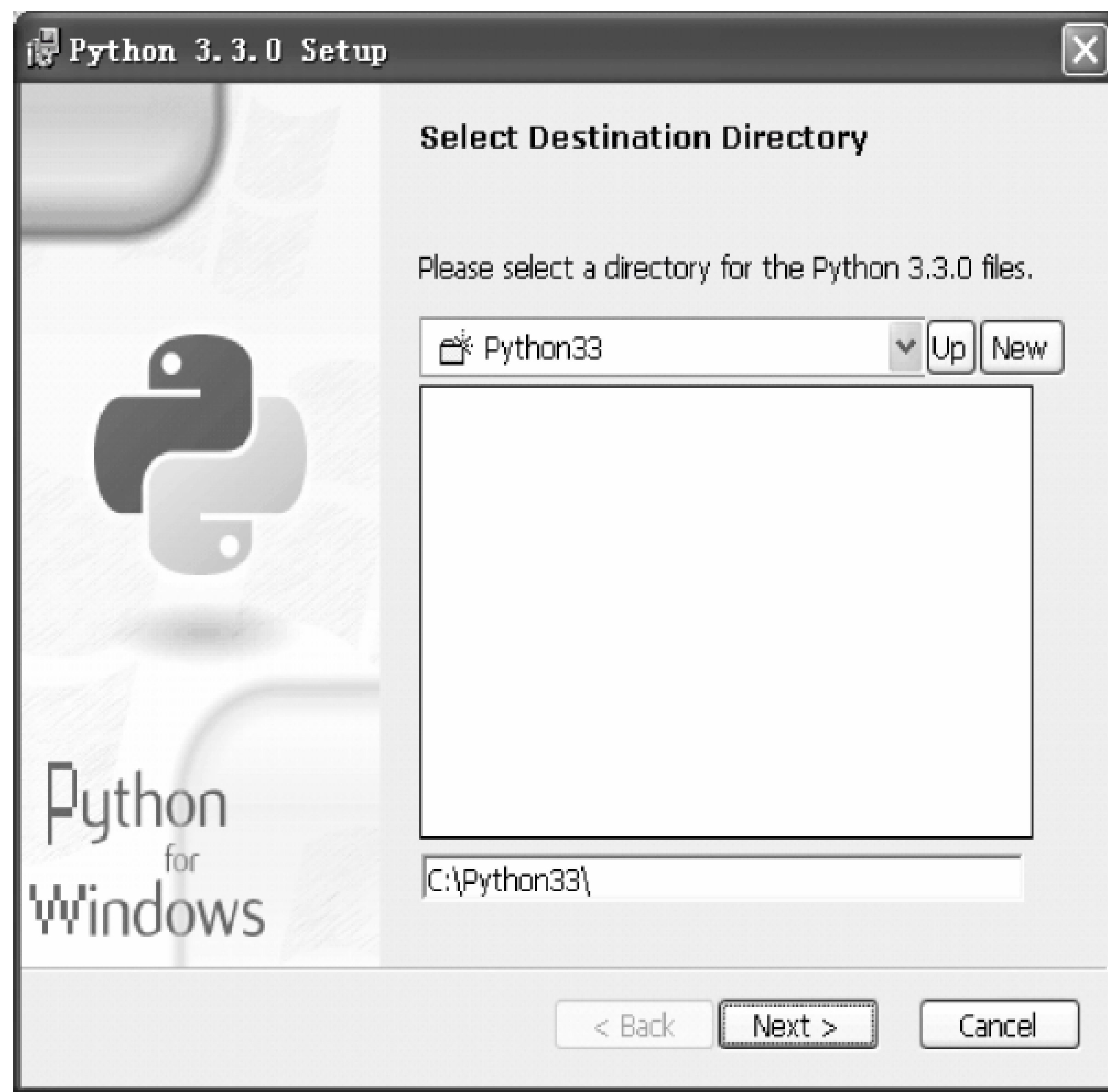


图 1-3-5　安装画面的路径选择

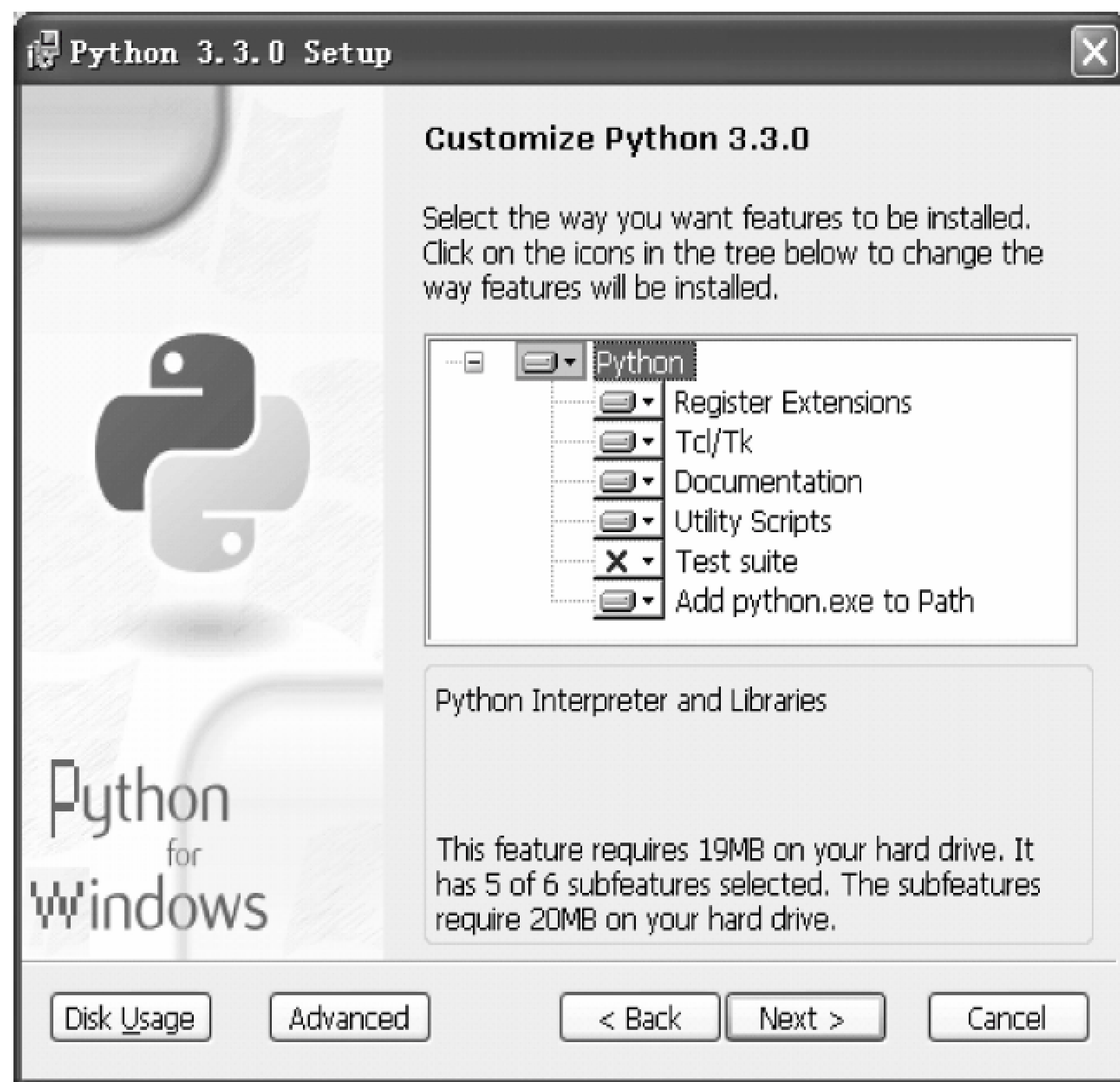


图 1-3-6　安装画面的自定义安装选项

在此画面中,可以自由选择需要安装的套件,如没有存储空间的限制,建议全部安装。然后单击 Next 按钮,弹出如图 1-3-7 所示的正在安装画面。

当安装结束的时候,将弹出如图 1-3-8 所示的安装成功画面,如果没有弹出该画面,请到官网或者相关论坛,查看原因。

单击 Finish 按钮,即完成 Python 的安装。

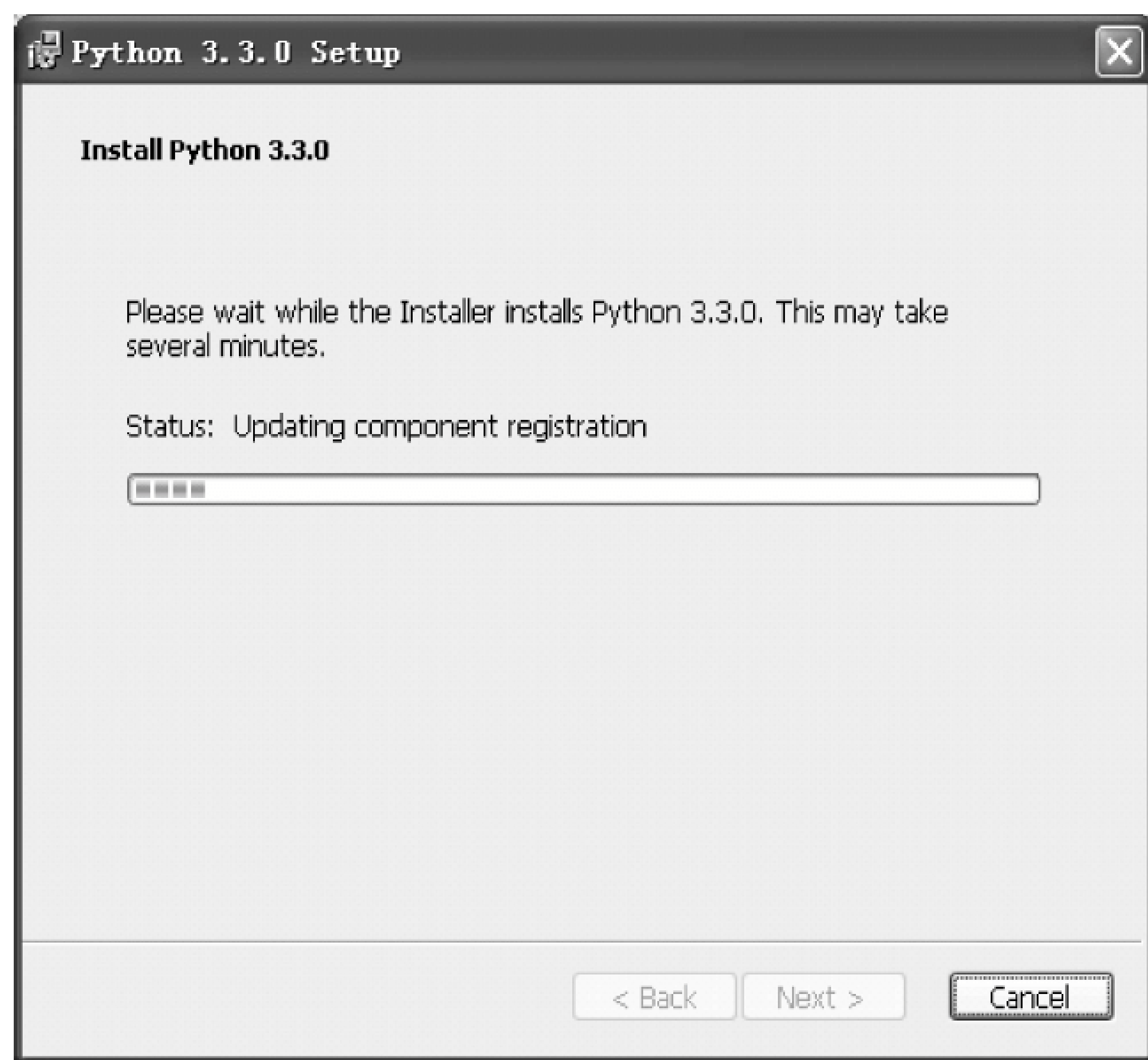


图 1-3-7　安装画面的正在安装画面



图 1-3-8　安装画面的结束画面

2. 运行 Python

若显示安装成功。选择"开始"→"所有程序"→Python3.3→IDLE(Python Integrated Development Environment),打开 IDLE 出现如图 1-3-9 所示的界面。

到此为止,最简单的 Python 开发环境就已经搭建好了。IDLE 是标准的 Python 开发环境,使用方便,其中最有用的就是查看 Python 文档,这对开发者来说极其方便,按 F1 键就能调出 API 文档。

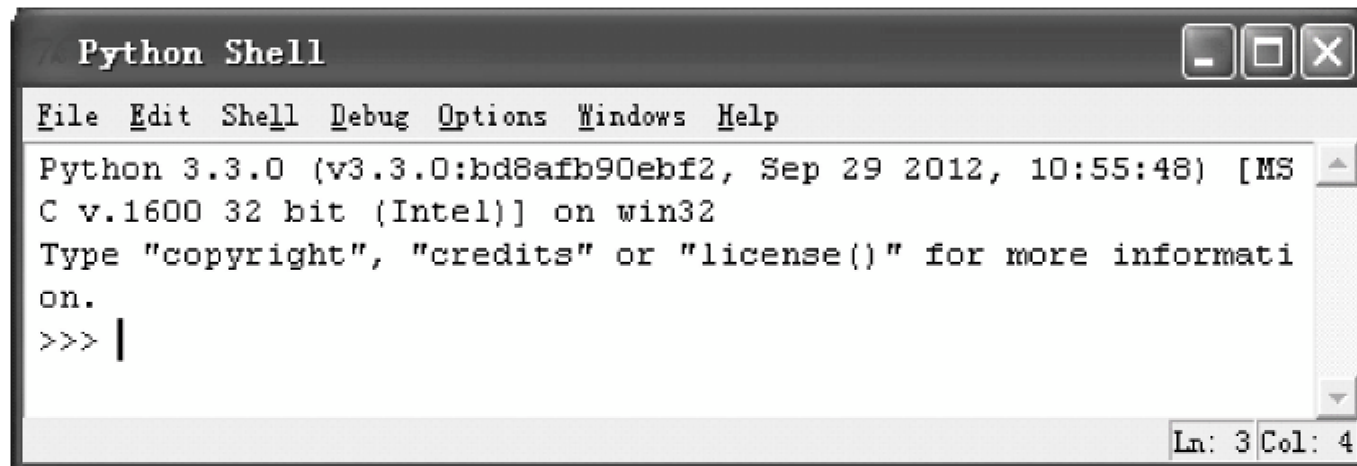


图 1-3-9　Python 的 IDLE 界面

3. 将 Python 当作计算器

启动 IDLE,等待主提示>>>出现。解释程序可以作为计算器使用。输入一个表达式,解释程序就可以输出结果。表达式的写法很直观: +,-,*,/,%,**等运算符的作用和其他大多数语言(如 PASCAL 或 C)没什么差别,括号可以用来组合。例如:

```
>>> 2 + 2
4
>>> # 这是一个注释
>>> 2 + 2 # 和代码在同一行的注释
4
>>> (50 - 5 * 6)/4
5.0
>>> 7/3
2.3333333333333335
>>> 7/ - 3
-2.3333333333333335
```

4. 体验 Python 中的哲学

在一本计算机语言的著作中讲述哲学,似乎有点偏题,但哲学是探求方法论的学科,它是指导人类思维的纲领,没有哲学就没有思维。

在 Python 的 IDE 环境中,只要输入 import this,就可以体验 Python 的设计哲学。这也是为什么把 import this 而不是 Hello 作为 Python 的第一个操作学习的内容。

5. 输出 Hello World

很多程序设计语言都从输出 Hello 或者 Hello World 开始程序设计学习之路。要实现这个程序,可以直接在 Python 的 IDLE 环境下输入 Python 语句运行,图 1-3-10 就是在 IDLE 中输入“print("Hello")”(按 Enter 键)后的显示结果。

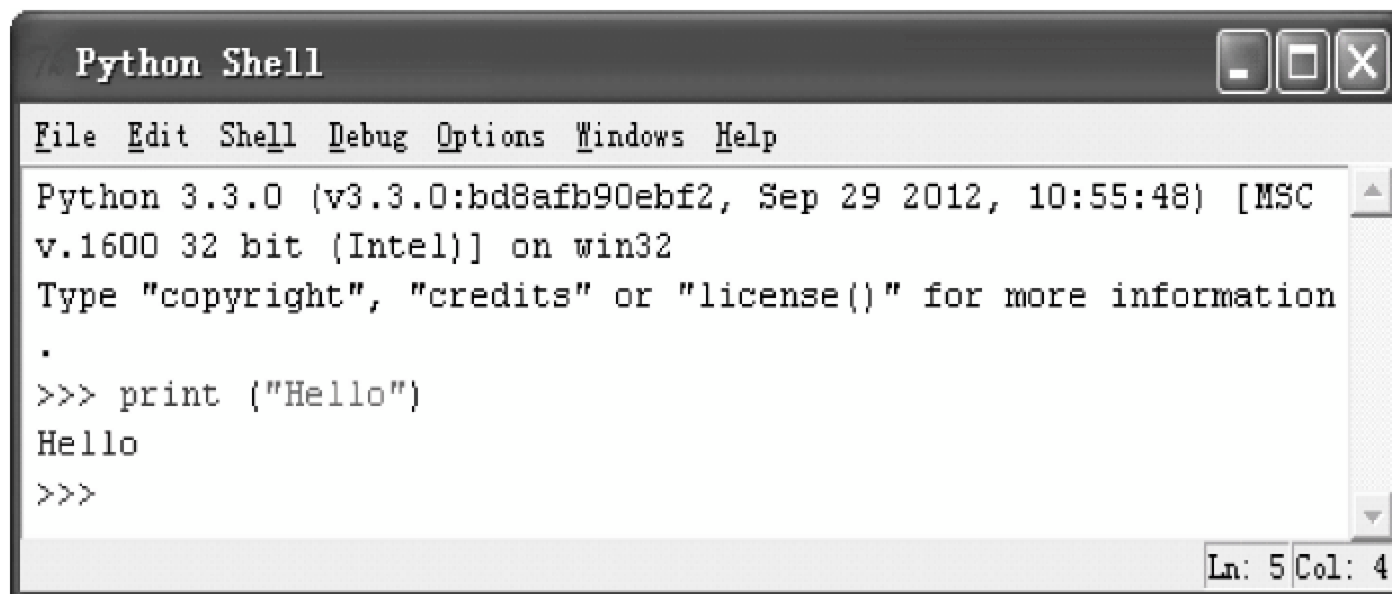


图 1-3-10　直接在 Python 的 IDLE 中运行语句

如果希望将该程序保存为 Python 的文件，可按下面的步骤操作：

选择 File→New Windows，在弹出的新窗口中，输入程序代码：print("Hello")。

单击菜单 File→Save，就可以将 Python 代码以文件的形式保存。

此处将文件保存为 Hello.py，py 是默认的 Python 代码文件的后缀。

Python 在执行时，首先会将.py 文件中的源代码编译成 Python 的 Byte Code（字节码），然后再由 Python Virtual Machine（Python 虚拟机）来执行这些编译好的 Byte Code。这种机制的基本思想跟 Java，.NET 是一致的。然而，Python Virtual Machine 与 Java 或.NET 的 Virtual Machine 不同的是，Python 的 Virtual Machine 是一种更高级的 Virtual Machine。这里的高级并不是通常意义上的高级，不是说 Python 的 Virtual Machine 比 Java 或.NET 的功能更强大，而是说和 Java 或.NET 相比，Python 的 Virtual Machine 距离真实机器的距离更远。或者可以这么说，Python 的 Virtual Machine 是一种抽象层次更高的 Virtual Machine。

基于 C 的 Python 编译出的字节码文件，通常是.pyc 格式。

除此之外，Python 还可以以交互模式运行，如主流操作系统 UNIX/Linux、Mac、Windows 都可以直接在命令模式下直接运行 Python 交互环境。直接下达操作指令即可实现交互操作。

6. 第二个程序 Input

第一个程序只是通过 print 语句，把 Hello 这个字符串输出来，通过第一个程序，读者应该掌握如何在 Python 语言中输出简单的信息，接下来，再通过第二个程序让读者掌握如何在 Python 语言中接收输入的信息。

选择 Python 的 IDLE 环境中的菜单 File→New Windows，在弹出的新窗口中，输入下述程序代码：

```
# ----------------------------------------------------------------
# This is my second Program
# ----------------------------------------------------------------
a = input("请输入一个数字: \n")
print(a)
print(a + a)
```

选择 File→Save，将 Python 代码以 input.py 文件保存。然后按 F5 键显示以下的运行结果：

```
>>>
请输入一个数字:
3
3
33
>>>
```

上述命令的执行结果可以看到，input 是类似于 print 的函数，它负责接收用户输入的信息，而 input 里跟的参数作为输入信息的提示内容。input 和 print 不同的是，它是有返回值的函数，它的返回值就是用户输入的信息。在上述代码中，input 函数将接收到的输入信息，存储到 a 这个变量中，然后再直接输出 a 和 $a+a$。因为屏幕上输入了一个数字 3，所以

输出 a 的时候输出了 3,但是在输出 $a+a$ 的时候却输出了 33,而不是 6,为什么呢？原因就在于不管输入什么内容,在通过 input 函数接收后,都是字符串类型的数据,如果有别的需要,在接收完后,必须进行数据类型转换。因为是字符型的数据,所以 $a+a$ 表达式中的加号就成了字符串拼接的意义,那 $a+a$ 也就变为 33 了。具体的细节将在第 3 章中介绍。

7. 关于 Python 的资源

为了帮助读者更好地学习 Python,下面罗列了很有帮助的一些在线 Python 资源：

- 中文的简明 Python 教程
  http://zhgdg.gitcafe.com/static/doc/byte_of_python.html
- 挑战智商的 Python 在线测试
  http://www.Pythonchallenge.com
- 编程趣味学习网站
  http://www.codecademy.com/zh
- Pygame 学习网站
  http://www.pygame.org

1.4 Python 的帮助系统

1.4.1 关于 Python 帮助系统

任何一个语言都有自己的示例程序和开发文档,这些内容都是为了用户能够更好地使用该语言。在语言的安装包中,这些内容也成为可选的安装项。虽然现在以百度知道、CSDN 为代表的各种论坛,都可以方便地查阅相关的知识,但是一个语言的帮助系统是该语言的第一手资料,是原生态的内容,比较有权威性,很多论坛的内容也大都摘抄自帮助系统,所以学会使用一个语言的帮助系统,可以让大家在学习语言的过程中事半功倍。

Python 语言的帮助系统是全英文的,对一些函数和语法介绍得比较详细,但是缺少太多的实例。毕竟与其看大段的文字,不如去研读一些实例显得更高效。

1.4.2 使用 Python 帮助系统

在 Python 的 IDE 环境中,只要按 F1 键,就可以显示帮助窗口,如图 1-4-1 所示。



图 1-4-1　帮助主窗口

该图是 Python 帮助系统的主界面，最上面的工具栏列举了一些比较常用的功能，如字体的选择可以使读者可以更方便地查看帮助的内容，打印可以帮助读者把一些重要的信息打印出来。如果比较系统的查看帮助，可以选择“目录”选项，如果查找特定的内容，建议使用“索引”。

例如，想要掌握 Print 函数的使用方法，可以在“索引”界面中输入 Print，弹出如图 1-4-2 所示的画面。

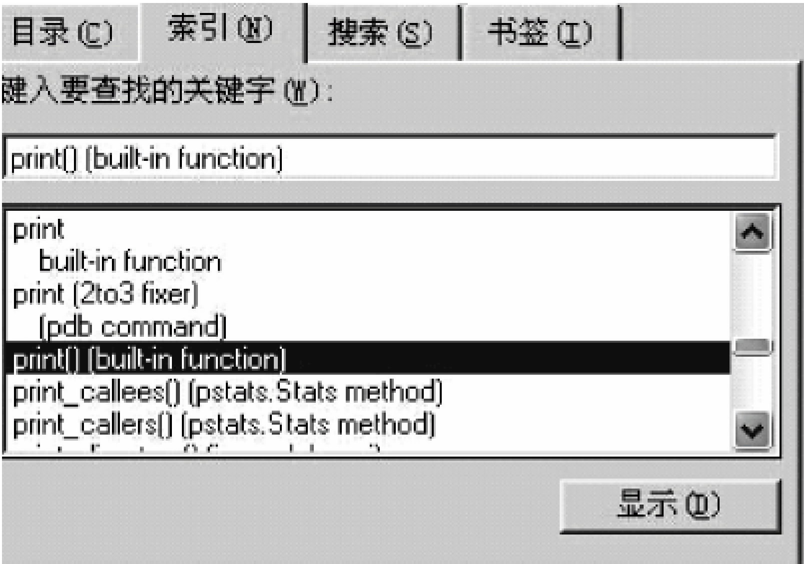
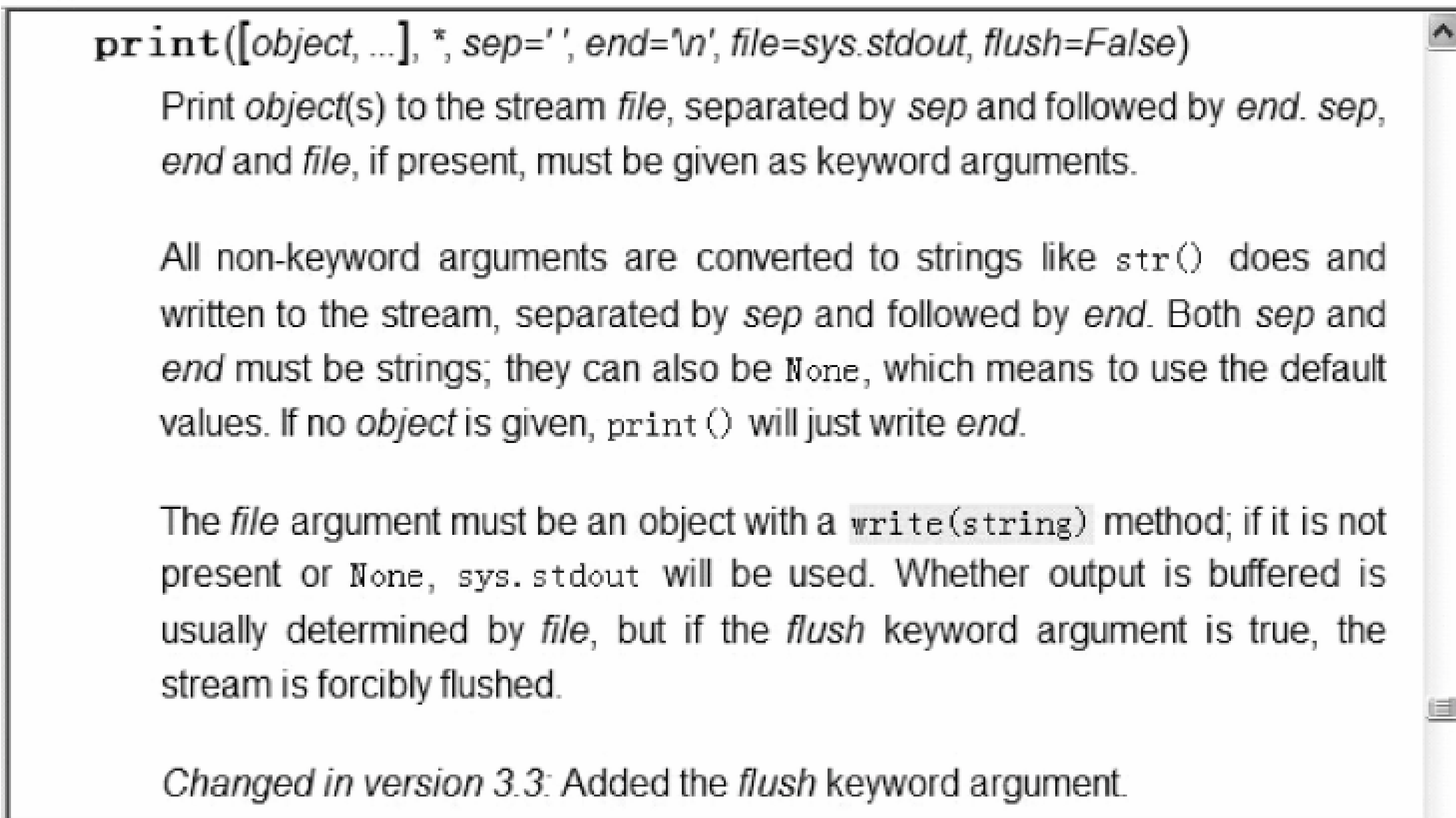


图 1-4-2　print 搜索结果

图中显示了所有关于 print 模糊匹配的条目，因为要查看它作为函数的使用方法，所以选择 print()(built-in function)条目，内容如图 1-4-3 所示。



print([*object*, ...], *, *sep*=' ', *end*='\n', *file*=sys.stdout, *flush*=False)

Print *object*(s) to the stream *file*, separated by *sep* and followed by *end*. *sep*, *end* and *file*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *object* is given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Whether output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

Changed in version 3.3: Added the *flush* keyword argument.

图 1-4-3　print 的具体内容

图中显示了 print 内置函数的使用方法，最后一行特别指出了新版本的变化。

综上便是 Python 帮助系统的使用方法。希望它能成为广大读者攻克编程语言堡垒的一个利器。

1.5　本 章 小 结

本章内容重点介绍了程序设计和计算思维概念以及它们之间的关系，并由此引出了本书所着重介绍的语言 Python。

本章要点如下：

(1) 程序设计的概述、分类以及设计步骤和规范。按照程序设计的成分性质分类，程序设计有顺序程序设计、并发程序设计、并行程序设计、分布式程序设计之分。

(2) 计算机语言是人与计算机之间传递信息的媒介。按分类，可以分成机器语言，汇编语言，高级语言三大类。

(3) 程序设计和计算思维的介绍。

(4) 计算思维与计算科学的关系。

(5) 计算思维与程序设计语言的关系。

(6) Python的概要和特点,它是一个在C语言基础上发展起来的支持面向对象、内存自动回收、支持动态类库和外接函数库、跨平台的开源高级语言,它的应用涉及网络编程、图形编程、数学应用、数据库应用、多媒体应用、Web编程等方面。它的语法简单而优雅,更加贴近自然语言,非常适合编程基础的初学者学习。

(7) Python的编辑和运行环境使用的是官方网站下载的IDLE环境。

(8) Python的交互式命令,以计算器和体验Python哲学为例进行了介绍。

(9) 编写Python程序时,应该注意的问题,如缩进、注释、如何运行程序等。

(10) Python的源代码文件,如何保存、保存格式。

(11) Python帮助系统的使用,以Print函数为例,进行了介绍。

1.6 习题与思考

1-1 以下________不是面向对象程序设计语言。

A. Java　　B. C语言　　C. Python　　D. C#

1-2 程序中的错误主要分为语法错误和________错误。

1-3 计算机语言的种类很多,可以分成机器语言,汇编语言,________三大类。

1-4 按照程序设计的成分性质分类,程序设计有顺序程序设计、并发程序设计、并行程序设计、________之分。

1-5 请读者讨论自己所学过的一些编程语言并简要说出Python的特点。

1-6 Python语言是由________语言发展而来的。

1-7 请罗列下Python语言的应用范围。

1-8 Python语言的官方网站是________。

1-9 Python编译出的字节码文件,通常是________格式。

1-10 Python语言接收信息的内置函数是________。

1-11 请读者利用Python的IDE环境,编写程序,输出自己的姓名。

1.7 实验 Python的安装和运行环境

1.7.1 实验目标

(1) 掌握Python的安装方法。

(2) 掌握Python IDE的基本操作。

1.7.2 实验范例

1. 体验Python设计哲学

(1) 打开Python的IDE环境。

(2) 在Python的IDE环境的主提示符后,输入import this,然后运行该命令,尝试将屏幕出现的英文内容,翻译为中文。

2. 使用Python的帮助系统

(1) 打开Python的IDE环境,然后按F1键打开帮助系统。

(2) 在索引中输入 input,按 Enter 键。

(3) 在随后出现的列表中,选择 input()(build in function)条目,查看 input 函数的使用说明。

1.7.3 实验内容

(1) 通过"开始"→"所有程序",启动 Python IDLE(Python GUI),首先查看 Python 版本信息。

(2) 在 Python IDLE 提示符下输入:help(),然后在 help 提示符下输入:print。了解 print 函数的语法。最后输入:quit,退出帮助界面。

(3) 在 Python IDLE 的 Help 菜单中选择 Python Docs 命令,或者直接按 F1 键进入系统帮助文档,通过"目录"或"索引"选项卡选择相关内容获取帮助信息。

(4) 选择 Python IDLE 的菜单 File→New Window 命令(或直接按 Ctrl+N 键),然后在空白窗口中输入:print("自己的姓名和学号"),再选择 File→Save 命令,保存为 me. py 文件(保存在 Python 安装文件夹下)。执行 Run→Run Module 菜单命令运行程序。最后执行 File→Close 菜单命令关闭程序窗口。

(5) 在 Python IDLE 的 File 菜单中选择 Open 菜单项,通过文件浏览对话框,打开 sy 文件夹下的 hello_world1. py,并按 F5 键运行该程序。本来应该通过该程序输出 Hello World,而程序却输出了 Hello Word,这就是逻辑错误。

(6) 在 Python IDLE 的 File 菜单中选择 Open 菜单项,通过文件浏览对话框,打开 sy 文件夹下的 hello_world2. py,并按 F5 键运行该程序。本来应该通过该程序输出 Hello World,而程序弹出了错误对话框,如图 1-7-1 所示。

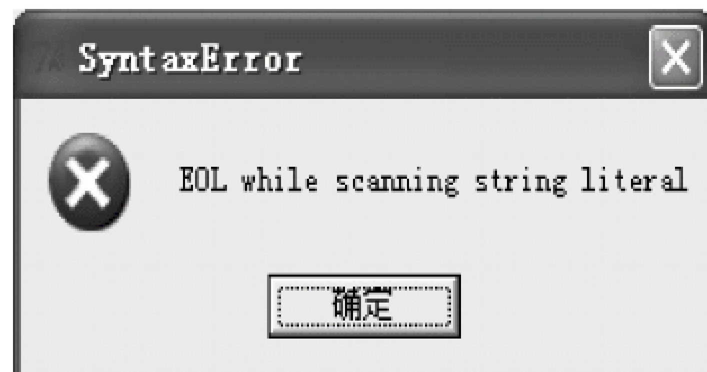


图 1-7-1 语法错误对话框

该对话框说明程序有语法错误,请检查该程序的语法错误。

第2章　算法概述

在进行程序设计时，首先必须考虑应该以何种方法或策略对实际问题进行求解。只有对问题求解的方法和大致步骤心中有数，才能保证后面的编程顺利进行，以避免走弯路、甚至推倒重来的尴尬局面。这就是算法的重要性。所以说，算法是程序的灵魂，程序语言只是实现算法的工具。

2.1　计算机程序与算法

所谓“算法”是指解决一个“计算”问题所采取的策略和过程。这个过程包括了具体的操作处理顺序。

以下通过日常生活中的一个例子来说明正确指定行动顺序的重要性：

【例2-1-1】　设计一个起床—上班的“算法”。

甲的“算法”为：

①起床。②整理床被。③洗漱。④换衣服。⑤烧早饭。⑥吃早餐。⑦出门上班。

乙的“算法”为：

①起床。②整理床被。③烧早饭。④洗漱。⑤吃早餐。⑥换衣服。⑦出门上班。

丙的“算法”为：

①起床。②整理床被。③吃早餐。④洗漱。⑤烧早饭。⑥换衣服。⑦出门上班。

在上述三个“算法”（行动顺序）中，甲显然是可行的，但由于乙在烧早饭的同时进行洗漱，减少了等待时间，所以效率更高，而且在吃完早餐后再换正装出门似乎更合理。对于丙来说，显然不符合生活常识，实际不可行。

计算机算法当然是指在计算机上进行数据处理和问题求解的方法和策略，但与上面的例子有着相同的性质。

2.1.1　计算机求解问题的过程

目前使用的冯·诺依曼结构计算机是采用程序存储与程序运行的原理，即计算机中的任何操作都必须事先编制程序，存入计算机的内存，然后运行程序完成指定的操作，而程序就是为完成指定任务所设计的计算机指令的有序集合。

程序中安排的计算机指令必须要有正确的执行顺序才能产生预定的结果。另一方面，由于计算机程序有极其严格的语法定义和结构形式，为了在解决问题之初能对问题的处理策略和过程有一个全局设计，并将注意力集中在问题主要方面，避免一些细节问题的缠扰，需要在具体编程前，先确定算法，并用相对简单的方法将处理策略和过程描述出来。

计算机程序开发的过程如图 2-1-1 所示。

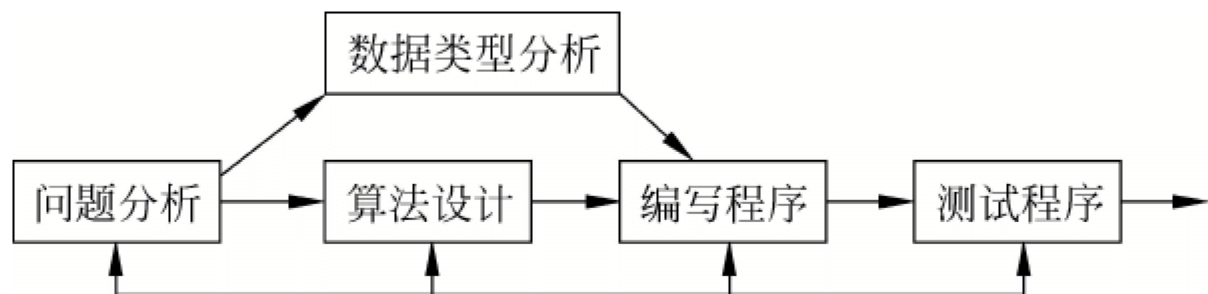


图 2-1-1　计算机程序开发过程

由图 2-1-1 可知,算法是问题的程序化解决方案。要使计算机能完成人们预定的工作,首先必须为如何完成该工作设计算法,然后再根据算法编写程序。

2.1.2　算法的定义及其发展历史

算法是从数学上演算法发展而来的。我国古代就有称为"术"的算法,最早出现在《周髀算经》和《九章算术》中。

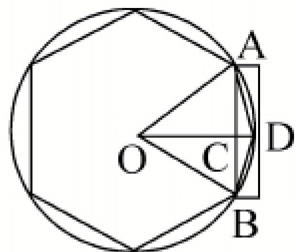
《周髀算经》约成书于公元前 1 世纪,是我国最古老的天文学著作,主要阐明当时天文学的盖天说和四分历法学说。《周髀算经》在数学上的主要成就是介绍了勾股定理及其在测量上的应用。

《九章算术》系统总结了战国、秦、汉时期的数学成就。它的出现标志中国古代数学形成了完整的体系。《九章算术》以计算为中心,以解决人们生产、生活中的数学问题为目的,给出了平面几何图形面积的计算方法、分数的四则运算、最大公约数、最小公倍数、开平方根、开立方根以及线性方程组求解等算法。

此后闻名于世的又有魏晋时代的刘徽(图 2-1-2(a))割圆术(求圆周率算法)(图 2-1-2(b)),以及唐代著名的杨辉算法等。



(a) 刘徽



(b) 刘徽割圆术示意图

图 2-1-2　中国古代"术"

算法的英文名 Algorithm 来自于 9 世纪波斯数学家花拉子米(比阿勒·霍瓦里松,波斯语:خوارزمی,拉丁转写:al-Khwarizmi)。因为比阿勒·霍瓦里松在其影响深远的著作《代数对话录》中提出了算法这个概念。al-Khwarizmi 意思就是"花拉子米"的运算法则。

欧几里得算法被公认为史上第一个算法。到 20 世纪,英国科学家图灵提出了著名的图灵论题,并提出一种假想的计算机抽象模型,称为图灵机。图灵机的出现对算法的发展起到了重要的作用。

从计算机的角度看,算法是计算机处理信息的本质,它告诉计算机按照确切的步骤来执行一个指定任务。通俗地说,算法是以一步一步的方式来详细描述如何将输入数据(或问题)转化为所要求的输出(问题结论)的过程。比较规范的说法是,算法是有限个步骤内求解某一问题所使用的一组定义明确的规则和方法,是对计算机上执行的计算过程的具体描述。

算法的核心是创建问题抽象的模型和明确求解目标,之后可以根据具体问题选择不同的模式和方法完成算法的设计。

计算机算法是程序的灵魂。先有算法,再有程序。当一个算法使用计算机程序语言描

述时就是程序。换一个角度,从计算机解题的过程看,无论是构建形式、解题思路还是编写程序,都是在实施某种算法:前者是推理实现算法,后者是操作实现算法。

2.1.3 算法的基本性质

算法必须具备以下性质:

- **正确性**:算法首先应该是正确的。即对于任意的一组输入(包括合理的输入与不合理的输入),总能得到预期的输出。

例如,在前面的例子中,根据丙的"算法",先吃早餐再烧早饭显然是错误的。

- **可行性**:由于算法是程序设计的依据,算法的每一步都要能够被计算机理解和执行,而不是抽象和模糊的概念(大概、近似、比较小等)。
- **确定性**:算法的每个步骤都有确定的执行顺序,每一步是什么,都必须明确,无二义性。
- **有限性**:算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。无论算法有多么复杂,都必须在有限步之后结束。计算机按照算法编制的程序,运行后能在有限的步骤内终止:或者终止于得到问题的解,或者得出问题无解的结论。
- **输入和输出**:一般地,当算法在处理信息时,需要从输入设备或数据的存储地址读取数据,经过"计算"后把结果写入输出设备或某个存储地址供以后再调用。所以一个算法有 n 个输入(特殊情况下也可以为零个),以说明运算对象的初始情况;算法中也应该有一个或多个输出,即最终结果(与输入有某个特定关系的量)。

例如,求两数的最大公约数算法,输入为所求的两个整数(m、n),输出为最大公约数。

前面提到算法的正确性时曾指出,一个良好的算法不仅能够对合理的输入数据进行处理、输出正确结果,对输入的非法数据也应该能做出恰当反应或进行相应处理。这又称为算法的"健壮性"。

2.1.4 算法的评价

通常,解决问题的途径可能会有多种。例如前面的起床—上班"算法"中,甲和乙都能完成此过程。计算机解决一个问题的算法同样不是唯一的,恰恰相反,很多步骤和思路完全不同的算法可以解决相同的问题。

对于同一问题的多种算法,有一个评价问题。评判一个算法的优劣可从算法执行时间(常用核心语句执行次数与问题规模 n 的函数关系量度)和要占用的额外空间两方面考虑,称为时间复杂度和空间复杂度。

有关算法性能分析将在第 7 章"算法分析与设计"中具体讨论。

2.2 算法的描述

算法的描述有多种方式,可以用流程图、伪代码等,甚至用人们日常所用的语言,如汉语、英语、德语等自然语言描述。

2.2.1 用自然语言或伪代码描述算法

自然语言不用专门训练，所描述的算法也通俗易懂。但由于需要根据算法编制程序，所以直接使用自然语言描述算法不够简单明了，与程序设计的关系也不够密切。

伪代码(Pseudocode)是一种算法描述语言，就是用简明扼要的文字以及类似数学上的表达式，按类似程序的格式写出的算法。它可以引入程序设计的形式结构(如顺序、分支、循环三大基本结构)，但并不能在编译器上通过编译(或解释)生成运行程序。

伪代码在结构上与程序比较接近。用伪代码书写的算法介于自然语言与程序设计语言之间。虽然有些教科书上给出了一些书写法则，但并无严格的语法规则，一般要求结构清晰，简洁易读，有助于按此思路编写出实际程序，以及方便别人阅读理解程序。

【例 2-2-1】 温度转换。

问题分析：预报摄氏温度，求对应华氏温度。

数据关系：华氏温度 F 和摄氏温度 C 两者的对应关系是 $F=(9/5)C+32$。

算法(伪代码形式)：

输入：摄氏温度 celsius。

计算：华氏温度 fahrenheit=(9/5)×celsius+32。

输出：显示 fahrenheit。

讨论：考虑程序的健壮性，要求能够判断输入的数据是否合理，并能剔除不合理的数据。所以算法中需要在输入数据后，进行数据的合法性判断及处理。

改进算法(伪代码形式)：

输入：摄氏温度 celsius。

判断：输入的温度在合理范围，转下一步计算；否则，转上一步输入重新输入。

计算：华氏温度 fahrenheit=(9/5)×celsius+32。

输出：显示 fahrenheit。

【例 2-2-2】 选最大数。

问题：有一串随机数列，要找到这个数列中最大的数。

分析：如果将数列中的每一个数字看成一颗豆子的大小，可以使用下面的“捡豆子”算法。

① 将第一颗豆子放入碟中。

② 取第二颗豆子开始检查：如果正在检查的豆子比碟子中的大，将它捡起放入碟子中，同时丢掉原先碟中的豆子，反之则舍去该豆子。

③ 继续取下一颗豆子，重复第②步，直到最后一颗豆子。

④ 最后留在碟中的豆子就是所有豆子中最大的一颗。

注：本算法中包括了程序设计的基本结构：分支(步骤②)和循环(步骤③)。

2.2.2 用流程图描述算法

使用自然语言或伪代码描述算法不够简单明了，甚至容易产生歧义。对于一些简单问题尚能胜任，但遇到稍复杂的问题往往显得头绪纷繁，使人无从下手。

实际使用中常用图形方法来描述和表示算法，其中流程图就是一种最常用的描述算法

工具(需要指出的是,流程图使用范围很广,如数据流程图、业务流程图等,这里主要讨论用于描述算法的程序流程图)。

流程图使用一些指定的图框表示各种操作,并用带箭头的直线连接各图框,以描述它们之间的逻辑关系,不仅直观形象、易于理解,学习、掌握也很方便。

图 2-2-1 为美国国家标准化协会 ANSI 规定的常用流程图符号,已为各国普遍采用。

关于流程图常用图形符号的说明如下:

- 圆角矩形表示"开始"与"结束"。
- 平行四边形表示输入输出。
- 矩形表示操作处理。
- 菱形表示问题判断或判定(菱形的上角点为入口,其余角为出口。出口处常用 T(TRUE)/F(FALSE),或 Y(YES)/N(NO)表示判断结果的操作流向,也可写其他文字)。
- 箭头代表控制或操作流方向。
- 如果换页,通常在前一页的最后及后一页的开始处使用圆形连接符(在圆圈中写上相同的数字以便前后对应),也可用在同一页中以避免流程线过长或者交叉。
- 图中最后一行用于表示一段可能被多处重复使用的操作(过程)。

【例 2-2-3】 使用流程图表示例 2-2-2 选最大数的算法。

绘制的流程图如图 2-2-2 所示。

起止框	
输入/输出框	
处理框	
判断框	
流程线	
连接符	
过程调用	

图 2-2-1 常用流程图符号

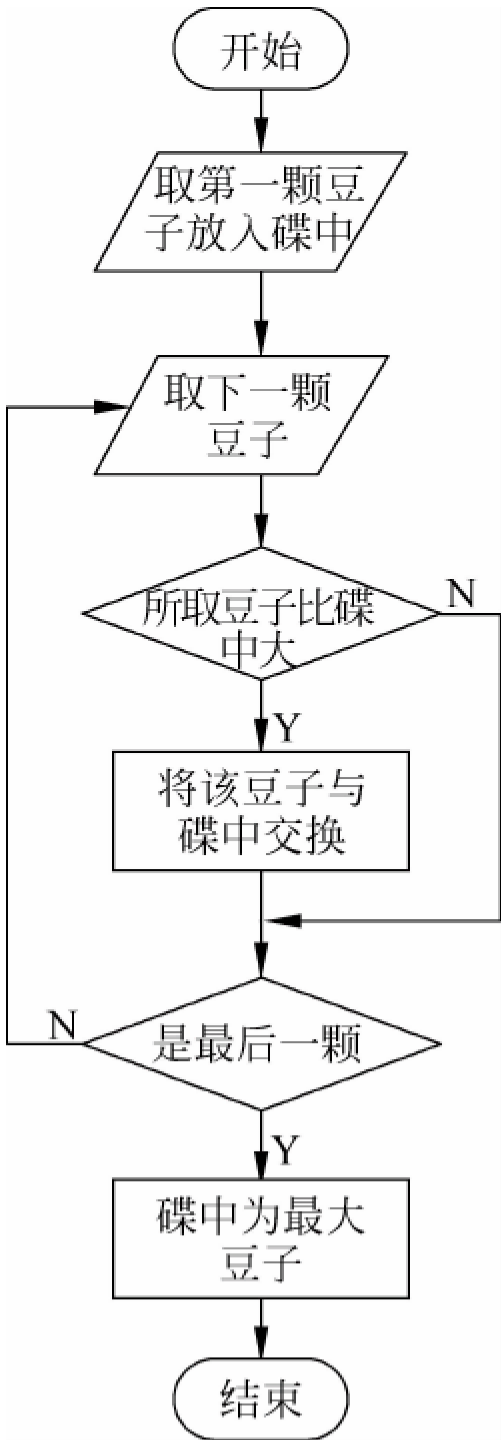


图 2-2-2 选最大数流程图

2.2.3 使用电脑软件绘制流程图

可以直接用笔和纸绘制流程图，也可以在电脑中使用文本编辑工具（如 Word）或绘图软件绘制流程图。

目前已经有多款专门适用于画流程图的软件，如 Microsoft Office Visio、OmniGraffle（运行在 Mac OS X 和 iPad 平台之上）和 ProcessOn 等。

ProcessOn 是一个面向流程用户的专业社交网站，成立于 2011 年 6 月，并于 2012 年启动。它为全球商业组织和个人提供一个共享的流程知识仓库，将结构化的流程实践给互联网用户分享。ProcessOn 的使用非常简单，用户只需通过注册便可获得这一永久免费的服务，并通过关注感兴趣的流程标签、专家和公司，可动态获取实用交流信息。

ProcessOn 网址为：http://www.processon.com，图 2-2-3 为输入该网址后的页面。



图 2-2-3　ProcessOn 网站

以下介绍使用 ProcessOn 绘制流程图（图 2-2-4）的方法。

① 登录后，单击右上角的“新建文件”按钮。

② 在弹出框中选择流程图分类及相关模板，缺省为“未分类”、“空模板”。单击“Flowchart 流程图”及“空模板”，单击“确定”按钮，然后在文件名对话框中输入文件名，这样就可以在弹出的画布上作图了。

③ 在左边的图形符号集工具栏中选择一个合适的符号，拖到画布中央。这样流程图的第一个图形就画好了。

④ 将鼠标移动到第一个图形的边缘，此时鼠标会变为十字形状。按住鼠标拖动，就拉出了一根带箭头的连接线。

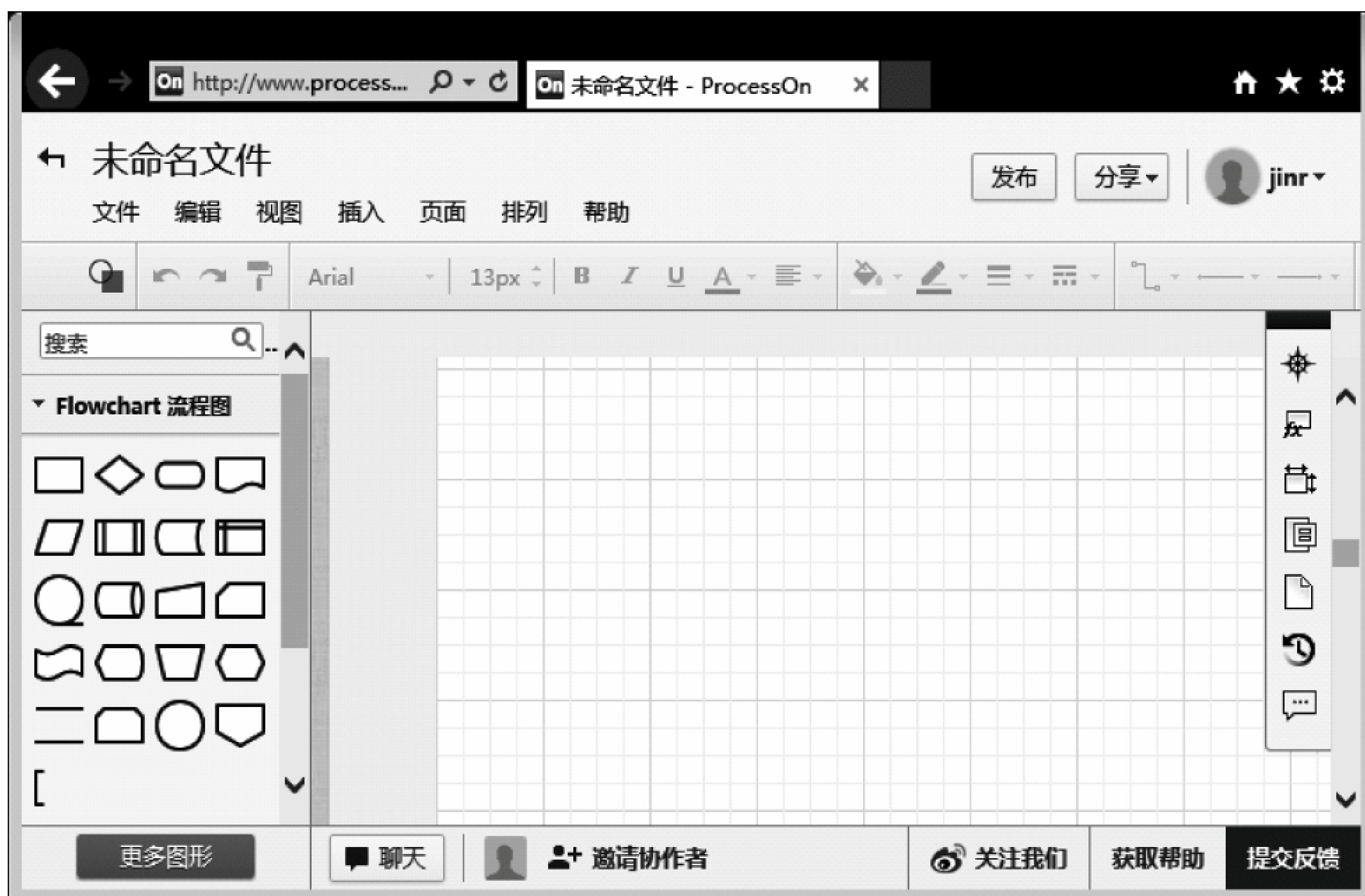


图 2-2-4　在 ProcessOn 中画流程图

⑤ 将鼠标松开,会自动弹出与刚才所画图形同一类型的图形列表,从中选择一个图形,则第二个图形就画好了,而且与第一个图形自动建立了连接。

⑥ 如果第二个图形与第一个图形不是同一类型的,只需在弹出图形列表时鼠标单击画布的其他任意位置,然后到工具栏拖曳所需类型的图形到画布上。

⑦ 双击图框(或右击图框,从快捷菜单中选择“编辑文本”),在图框中输入文字,并可通过菜单下面的工具栏设置字体、字号等文本格式。双击连接线也可在线上添加文字。

⑧ 重复以上操作完成流程图。

⑨ 选中某图框,当光标移到其角点处变为双向箭头时,可拖拽改变图框大小。当光标变为带 4 个方向箭头的形状时可拖曳移动图框位置。

⑩ 如果从上一个图形拉出的连接线没有与下一个图形相连,可选中连接线并将鼠标移动到其头部,当鼠标变为带 4 个方向箭头的形状时,按住鼠标拖曳到所需相连的图框,即可完成连接。

⑪ 根据需要,还可以对流程图图形及连接线进行填充颜色、边框颜色、连接线类型等的设置。

⑫ 流程图全部完成后,通过“文件”菜单中的“重命名文件”指定文件名保存;通过“文件”菜单中的“下载为”,可下载到本地计算机中,保存为 PNG 图形或 PDF 文件。

以上介绍了在 ProcessOn 中绘制流程图的基本方法。方法虽然简单,但也需要用户通过具体实践才能掌握。

最后需要指出的是,除流程图外,还有其他多种表示算法的图形,如 N-S 图、PAD 图(问题分析图)等,它们各有特点,甚至可弥补流程图的一些不足,限于篇幅这里就不介绍了。

2.3 常用算法简介

实际工程中会遇到许多复杂的计算问题，有的问题若采用劣质算法求解，即便在巨型计算机上可能也要花费数个月时间，但采用优秀算法，可能在一台普通计算机上仅需几分钟甚至数十秒就能解决。所以算法设计的优劣，对于问题的求解非常重要。

可以将所有算法粗分为以下两类：

(1) 简单的算法。

- 一般由现成公式作为算法描述。
- 算法通过一次性计算解决问题。

例如在工程学、力学、数学等学科中，

$G = g \times m1 \times m2/R2$（万有引力）

电压 = 电阻 × 电流（欧姆定律）

浮力 = 液体密度 × 物体排开液体的体积

圆的面积 = 圆周率 × 半径 × 半径

圆柱体体积 = 底面积 × 高

- 输入和输出描述。

对于简单算法，一般只需按“输入—处理—输出”（即所谓 IPO）的流程处理，如前面的摄氏-华氏温度转换算法。

(2) 复杂的算法。

- 一般没有终极公式作为问题解法。
- 可通过一次次反复“计算”，逐步得到问题的解或近似解。
- 每个计算周期“计算”问题的一部分或得到一个更好的近似解。

对于复杂算法，实际上是用简单问题的解逐步逼近原始问题解，适合于计算机求解。

在长期的实践中，前人已经留下了许多经典的算法。学习、掌握这些典型算法的思想，有助于深入了解计算机处理问题的思路，提高自己的计算思维能力，为今后使用计算机开发本专业的算法及应用程序打下良好的基础。

本章介绍几个基本的算法类型，后面还会介绍其他一些算法类型。

2.3.1 枚举算法

枚举法就是按问题本身的性质，一一列举出该问题所有可能的解，并在逐一列举的过程中，检验每个可能解是否真正满足问题所求。若是则采纳这个解，否则抛弃它。在列举的过程中，既不能遗漏也不应重复。

利用计算机快速运算的特点，通过构造循环结构并辅以判断语句可实现枚举算法，完成问题的计算。

1. 枚举算法的基本思想

- 根据问题描述和相关的知识，能为该问题确定一个可能的解空间范围。
- 枚举法就是对该解空间范围内的众多候选解按某种顺序进行逐一枚举和检验，直到找到一个或全部符合条件的解为止。

2. 应用枚举算法的通常步骤

(1) 建立问题的数学模型,确定问题的可能解的集合(解空间)。

(2) 确定合理的筛选条件,用于选出问题的解。

(3) 确定搜索策略,逐一枚举可能解集合中的元素,验证是否是问题的解。

3. 实现枚举算法的程序总体框架

枚举算法程序一般使用循环结构来实现,其总体框架如下:

设解的个数 n 初始为 0;
循环(枚举每一可能解):
　　检验:若(该解满足约束):
　　　　　输出这个解;
　　　　　解的数量 n 加 1;

计算机程序实现枚举算法的基本方法是:用循环结构实现一一列举的过程,用分支结构实现检验的过程。

4. 枚举算法的输入输出处理

- 输入:大部分情况下是利用循环变量来实现的,也可通过数组(列表)或集合的形式构成可能解的集合。
- 输出:一般情况下是在判断的一个分支中实现的,即满足条件的解即时输出。也可将满足条件的解存入一个数组(列表)或集合,待循环结束后统一输出。

【例 2-3-1】 求 1～1000 中,能被 3 整除的数。

算法(自然语言形式):

① 从 1～1000 中一一列举,这是一个循环结构。

② 在循环中对每个数进行检验:凡是能被 3 整除的数,打印输出,否则继续下一个数。这是一个分支结构。

【例 2-3-2】 判断谁是小偷。

某地失窃,有四个嫌疑人,分别谈话得到以下回答:

a 说:“我不是小偷。”

b 说:“c 是小偷。”

c 说:“小偷肯定是 d。”

d 说:“c 冤枉人!”

四人中有三人说的是真话,问到底谁是小偷?

说明:对本题这类非数值问题,经过数字化后,也可以用计算机程序进行处理。

算法分析:

使用枚举法,依次假设 a、b、c、d 为小偷,判断四人说话的真假。若在某假设下,得到的结果为三人说真话、一人说假话,则该假设成立,此即为所求的解。

例如,假设 a 是小偷,则根据以上四人的回答可判断 a、b、c 都说的假话,所以这种假设不成立;同样,假设 b 是小偷也不成立;假设 c 是小偷,则根据四人的回答可判断 a、b 和 d 都说的真话,只有 c 一人说假话,所以假设成立。

对 a、b、c、d 依次进行判断,显然这是一个循环的过程。为了能进行循环处理,需要对问题进行数字化。

- 设 x 为假设的小偷，x 依次取值 1、2、3、4(表示 a、b、c、d)；
- 用 $s1,s2,s3,s4$ 表示在某个假设下(即 x 分别为 1、2、3、4 时)，四人说话的状态(说真话为 1，说假话为 0)。

据此可得到以下算法。

初始：设 $s1=s2=s3=s4=0$　# =表示赋值

循环：对 $x=1,\cdots,4$　　　# 假设 4 个可能的解

在此假设下，分别求四人的说话状态($s1\sim s4$)。

若 $x!=1$，$s1=1$(a 说："我不是小偷。"真话)；# 用!=表示不等于

若 $x==3$，$s2=1$ (b 说："c 是小偷。"真话)； # 用==表示等于

若 $x==4$，$s3=1$ (c 说："小偷肯定是 d。"真话)；

若 $x!=4$，$s4=1$ (d 说："c 冤枉人!"真话)。

若经以上判断后有 $s1+s2+s3+s4==3$ (即三人说真话，一人说假话)，则假设成立，小偷即为此时 x 的值对应的那个人。

为了帮助读者熟悉、掌握流程图的画法，本例给出算法流程图，如图 2-3-1 所示。

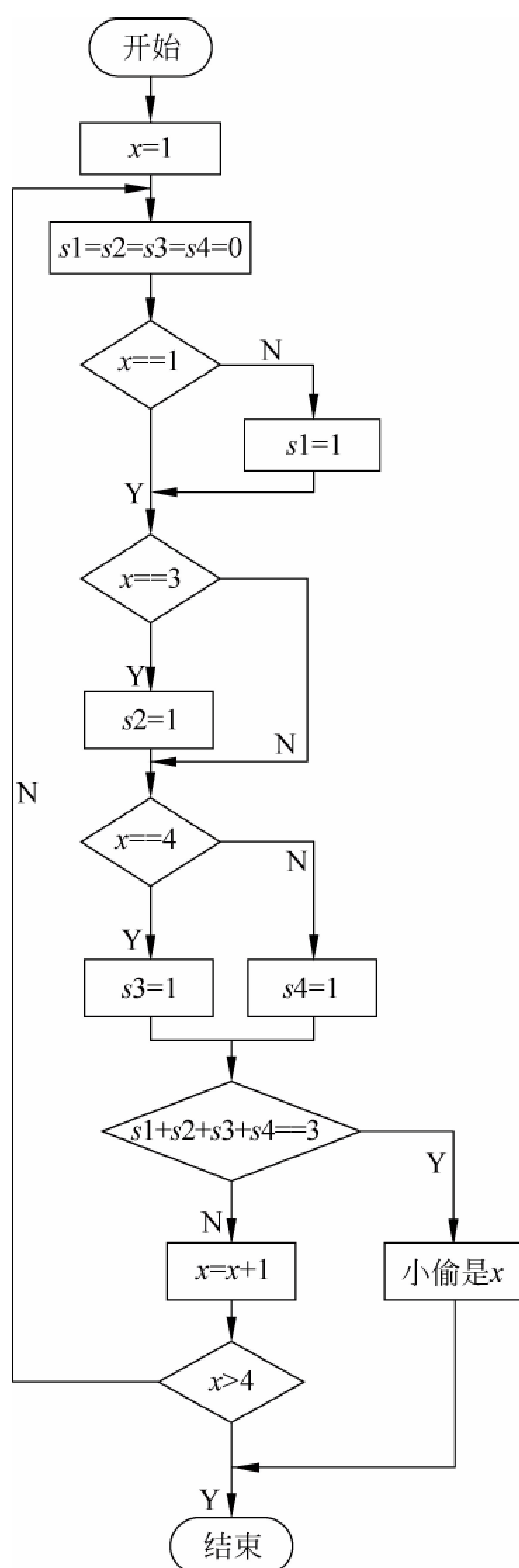


图 2-3-1　例 2-3-2 算法流程图

【例 2-3-3】 将苹果、橘子、香蕉、梨、菠萝 5 种水果做成水果拼盘，每个水果拼盘有三种不同的水果，且有序摆放，问可以有多少种?

算法分析如下。

- 初始：使用列表保存 5 种水果名。
- 通过三重循环结构，枚举各种可能的方案。
  - ➢ x、y、z 的取值范围为 5 种水果(解空间)。
  - ➢ 它们互不相等，且摆放先后次序有区别(筛选条件)。
  - ➢ 输出所有可能的方案。

算法(伪代码形式)如下：

① 建立水果列表 fruit。

　② 使变量 x 遍历 fruit。

　　③ 对于 x 的每个值，使变量 y 遍历 fruit。

　　　④ 对于 x、y 的每个值，使变量 z 遍历 fruit。

　　　　⑤ 若 $z\neq x$ 且 $z\neq y$ 且 $x\neq y$，打印该方案。

5. 枚举算法的优化

通过一些已知的制约条件，减少解空间可能解的数量，使检验工作量较少，加快算法的执行速度。

例如，在例 2-3-3"水果拼盘"算法中，如果已有 $x==y$，则不必再进行对 z 的判断，可做以下改进：

① 建立水果列表 fruit。

　② 使变量 x 遍历 fruit。

③ 对于 x 的每一取值,使变量 y 遍历 fruit。

④ 若 $x \neq y$,使变量 z 遍历 fruit。

⑤ 若 $z \neq x$ 且 $z \neq y$,打印该方案。

2.3.2 迭代算法

在算法和程序设计中,迭代也是最常用的一种方法。简而言之,迭代是一种不断用变量的旧值递推出新值的过程。

1. 迭代算法的基本思想

迭代算法是计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点,从一个初始变量值出发,让计算机对一组指令(或一定步骤)进行重复执行,每次执行这组指令(或步骤)时,都从变量的原值推出它的一个新值,最终得到所求解。

举一个最简单的例子,考虑求 1+2+3+…,直到和达到 1000 时的自然数问题。可以设一个累加变量 s(初始 $s=0$),然后通过循环将自然数 1、2、3……依次加到该累加变量 s 中,直到 $s \geqslant 1000$ 为止,这时最后一个加入的自然数就是所求之数。这个例子就体现了迭代的思想,累加变量 s 就是迭代变量。

2. 应用迭代算法的通常步骤

利用迭代算法解决问题,一般需要以下步骤:

1) 确定迭代变量

在可以用迭代算法解决的问题中,至少存在一个(也可能有多个)直接或间接地不断由旧值递推出新值的变量,这个(些)变量就是迭代变量。

一般在确定迭代变量的同时,还要指定其初始值。

2) 建立迭代关系式

所谓迭代关系式,是指如何从变量的前一个值推出其下一个值的公式(或关系)。迭代关系式的建立是解决迭代问题的关键,通常可以用顺推或倒推的方法来完成。

3) 对迭代过程进行控制

什么时候结束迭代过程?这是编写迭代程序必须考虑的问题。根据算法“有限性”的性质,不能让迭代过程无休止地重复执行下去。

迭代过程的控制通常有两种情况:一种是所需的迭代次数确定,可以计算出来(例如求 1+2+3+…+100 之和);另一种是所需的迭代次数无法确定,需根据每次的迭代结果决定是否再次进行迭代(例如上面举的求 1+2+3+…,直到和达到 1000 时的自然数的例子)。对于前一种情况,可构建一个固定次数的循环来对迭代过程进行控制(常用程序设计中的 for 语句实现);对于后一种情况,需要在每次循环结束后判断是否已满足要求,以决定是否结束迭代过程(常用程序设计中的 while 语句实现)。

【例 2-3-4】 验证谷角猜想。

日本数学家谷角静夫在研究自然数时发现一个奇怪现象:对于任意一个自然数 n,若 n 为偶数,则将其除以 2;若 n 为奇数,则将其乘以 3 然后再加 1 。如此经过有限次运算后,总可以得到自然数 1。请写出验证该猜想的算法。

算法分析:

定义迭代变量为 n,按照谷角猜想的内容,可以得到两种情况下的迭代关系式。

当 n 为偶数时，$n=n/2$；

当 n 为奇数时，$n=n\times3+1$。

这就是需要计算机重复执行的迭代过程。

确定结束迭代的条件：分析题目要求不难看出，对任意给定的一个自然数 n，只要经过有限次运算后能够得到自然数 1，就完成了验证工作。因此，用来结束迭代过程的条件为：$n==1$。

本例迭代变量和迭代次数控制都由同一个变量承担。算法流程图如图 2-3-2 所示。

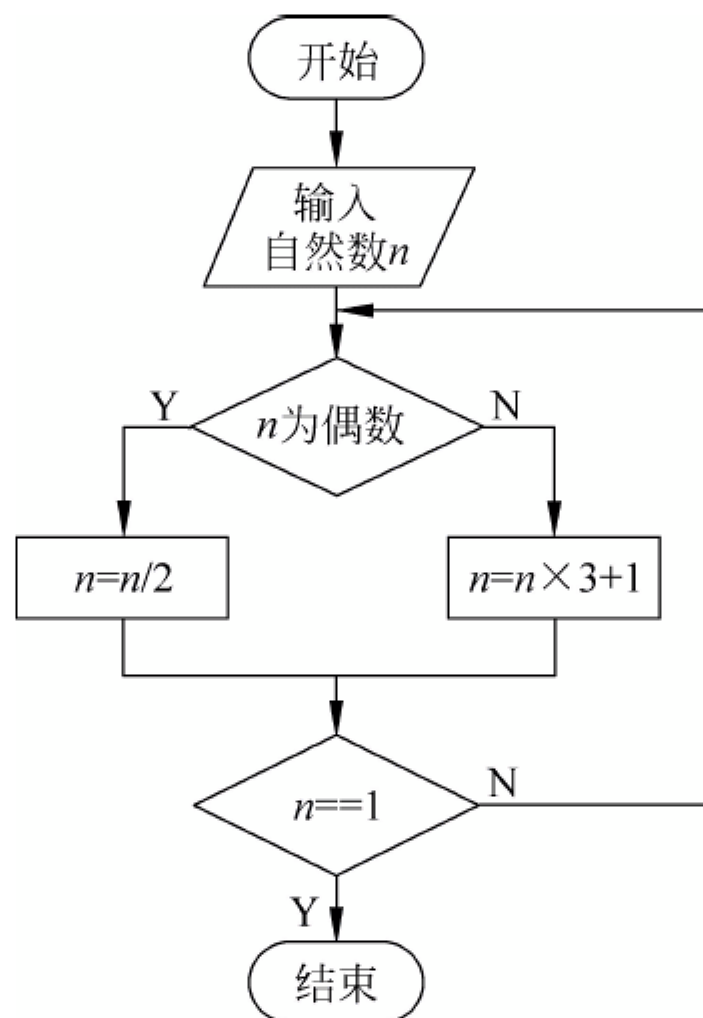


图 2-3-2 验证谷角猜想流程图

【例 2-3-5】 求 Fibonacci 数列前 40 个数。

Fibonacci 数列是一个意大利商人编写的养兔子模型。假定一对兔子从出生后的第 3 个月开始，每个月都繁殖一对(雌雄)兔子。如果没有死亡和出售的话，问到某月(此题为第 40 个月)后共有多少对兔子？

分析：这是一个典型的递推问题。不妨假设第 1 个月时兔子为 $R1$ 对，第 2 个月时兔子为 $R2$ 对，第 3 个月时兔子为 $R3$ 对……根据题意，"兔子从出生后的第 3 个月开始，每月新生一对兔子"，则有以下生成规律：

$$R1=1, R2=1, R3=R2+R1\times1=R2+R1, \cdots, Rn=Rn-1+Rn-2 \quad (n\geqslant3)$$

对应 Rn 和 $Rn-1$，定义两个迭代变量 y 和 x，再定义一个中间变量 z，可将上面的递推公式转换成以下迭代关系：$z=x+y, x=y, y=z$。

让计算机对这个迭代关系重复执行 38 次(初始 $x=1, y=1$)，就可以算出第 40 个月时的兔子数。

算法(伪代码形式)如下：

① 初始 $x=1, y=1$。

② 输出 x、y。

③ 循环：变量 i 从 3 到 40。

  ④ $z=x+y, x=y, y=z$。

  ⑤ 输出 z。

改进算法(舍去中间变量，每次输出两个值)如下：

① 初始 $x=1, y=1$。

② 循环：变量 i 从 1 到 20。

  ③ 输出 x、y。

  ④ $x=x+y, y=y+x$。

在改进算法中，因为开始两个月的数据尚未输出，所以进入循环后要先输出显示，再进行迭代。请读者自行分析该算法中对 x、y 赋值语句完成的迭代过程。

3. 递推与迭代

与迭代相近的概念是递推。设要求问题规模为 n 的解，当 $n=1$ 时，解已知或能方便得到，根据递推关系，能从 $i-1$ 规模的解，推出 i 规模的解($i=2,3,\cdots,n$)，这就是递推。

当然也可以反过来,从 $n,n-1,\cdots,3,2$ 反推到 1。

递推的过程实际上就是迭代的过程,即不断用变量的旧值推出新值的过程,或者说是根据前一个值推出后一个值。

在计算机程序设计中,一般递推使用数组(列表),在循环处理时利用其下标的变化实现变量的迭代,而狭义的迭代是指使用简单变量来完成这一过程。

程序设计中的数组(列表)是指具有相同名称、通过下标区分的一组变量,如 $a[0]$、$a[1]$、$a[2]$或 $b[1,1]$、$b[1,2]$、$b[1,3]$、$b[2,1]$、$b[2,2]$、$b[2,3]$等。在循环结构中,可通过变量控制其下标值的变化(如 $a[i]$、$b[i,j]$),达到变量轮换的目的。

【例 2-3-6】 植树问题。

植树节那天有 5 位同学参加了植树活动,他们完成植树的棵数都不相同。问第一位同学植了多少棵时,他指着旁边的第二位同学说比他多植了两棵;追问第二位同学,他又说比第三位同学多植了两棵……如此,都说比另一位同学多植两棵。最后问到第五位同学时,他说自己植了 10 棵。

问到底第一位同学植了多少棵树?

分析:设第 i 位同学植树的棵数为 a_i,欲求 a_1,需从第五位同学植树的棵数 a_5 入手,根据"多两棵"这个规律,按照倒序逐步进行推算:

(1) $a_5=10$。

(2) $a_4=a_5+2=12$。

(3) $a_3=a_4+2=14$。

(4) $a_2=a_3+2=16$。

(5) $a_1=a_2+2=18$。

这就是递推的思想。不难根据以上分析写出其算法。请读者自己尝试用伪代码写出来。

递推方法是数学解题中的一种常用方法。对一个序列来说,如果已知它的通项公式,则要求数列的第 n 项或前 n 项之和是很容易的。但许多情况下,要得到数列的通项公式很困难,然而可观察到数列相邻位置上的数据之间存在一定的关系。使用递推方法可以避开求通项公式的困难,在一次次循环中,通过已知的项推算出其后继值,直到最终找到所需的那一项。

【例 2-3-7】 杨辉三角形是揭示二项展开式各项系数的数字三角形,求解杨辉三角形,如图 2-3-3 所示。

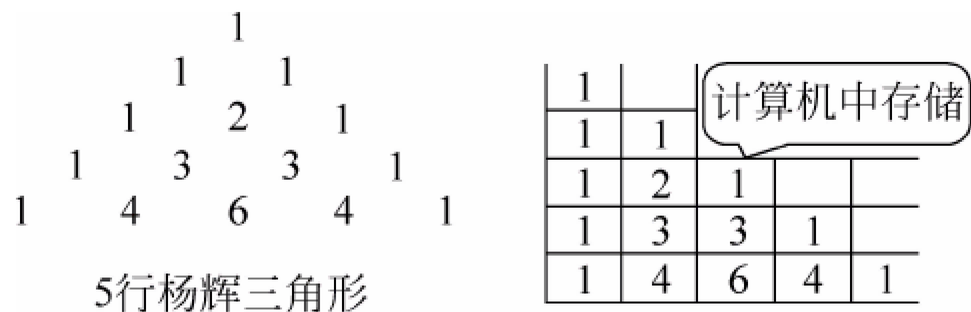


图 2-3-3 求解杨辉三角形

分析特点如下:

- 第 i 行有 i 个数;
- 每行的首尾两数均为 1;

• 除首尾两数外,其余各项数为上一行两肩上的数之和。

算法(伪代码形式)如下:

① 输入 n(杨辉三角形的行数)。

② 设置二维数组 $a[n,n]$。

③ 循环(i 从 1 到 n)。

对于每行的首尾数,

$$a[i,1]=1, a[i,i]=1\text{(首尾两数均为 1)}$$

对于每行的余数,

$$a[i,j]=a[i-1,j-1]+a[i-1,j]\ \text{(}i\text{ 为行数,}j\text{ 为该行中的列位置)}$$

④ 输出:打印以上值(即二维数组 $a[n,n]$的值)。

为打印出三角形形式,在循环中需控制打印位置及换行,具体由程序设计语句控制。

2.3.3 贪心算法

买东西时,售货员常计算最少需要找多少张零钱,如图 2-3-4 所示,以便简化工作。



图 2-3-4 贪心算法

例如买某物品需要 34.5 元,顾客交给售货员 100 元整,按照现在的货币体系,则售货员最少需要找 4 张零钞(65.5 元):

50 元一张、10 元一张、5 元一张、5 角一张。

这就包含了贪心算法的基本思想:即首先考虑面值不超过 65.5 元的最大钱币(50 元),然后在剩下的 65.5−50=15.5(元)中,找到不超过该值的最大钱币(10 元),最后在剩下的 15.5−10=5.5(元)中先后找到最大钱币 5 元和 5 角。

1. 贪心算法的基本思想

贪心算法(又称贪婪算法)是指在问题求解时总是做出当前看最好的选择。也就是说,不从整体最优上加以考虑,它所做出的仅是在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解,但对范围相当广泛的许多问题它能产生整体最优解或者整体最优解的近似解。所以在实际问题处理中也是一种常用的算法策略。

贪心算法也可用于求解一些构造类问题。当应用枚举算法求解构造类问题较为复杂、数据量太大时,应用贪心策略常可快捷地得出所需的解。

2. 贪心算法的求解策略

(1) 建立数学模型来描述问题。

(2) 把求解的问题分成若干个子问题。

(3) 对每一子问题求解,得到子问题的局部最优解。

(4) 把子问题的局部最优解合成为原来问题的解。

【例 2-3-8】 最优装载问题。

有 n 个集装箱希望能装上一艘载重量为 C 的轮船,其中集装箱 i 的重量为 Wi。由于载重量的限制,这些集装箱不能全部装船。在装载体积不受限制的情况下,问最多能装载多少个集装箱。

分析：本题条件是载重受限、体积不限，希望装载个数最多。这类最优装载问题可用贪心算法求解。其贪心选择策略是：重量轻者优先装载。

算法(伪代码形式)如下：

设数组 w[ ]存储每个集装箱的重量，c 为船允许载重量，n 为集装箱个数。

① 对数组 w 由小到大排序(即重量轻者在前)。

② residual←c　　　//变量 residual 存储剩余载重量。

③ 循环(i 从 1 到 n)：

　　若 $w[i] \leqslant$ residual：

　　　　输出一个解 i

　　　　residual←residual－$w[i]$

　　否则：

　　　　跳出循环

【例 2-3-9】 活动安排问题。

设有 n 个活动的集合 $S=\{1,2,\cdots,n\}$，其中每个活动都要求使用同一资源(如演讲会场)，但同一时间内只有一个活动能使用这一资源。怎样安排才可以举行尽可能多的活动？

分析：这是区间调度问题。

每个活动都有使用的起始时间 bi 和结束时间 ei（显然 $bi \leqslant ei$）。对于活动 i 和 j，若 $bi \geqslant ej$ 或 $bj \geqslant ei$(即一个活动结束后另一个才开始)，则活动 i 与活动 j 相容。因此问题即为：求相容的最大活动集合，可用贪心算法有效求解。

数据模型：使用二元组记录每个活动的开始时间和结束时间。如(2,5)表示 2 时开始，5 时结束。

算法策略讨论：先直观分析，能否用以下策略。

- 先开始者先服务？-------×

反例，若有以下三个活动：(1,5)，(2,3)，(4,5)。该三个活动按此策略选了第一个后，其余活动与之冲突，故不是最合适的方法。

- 短活动者优先？-------×

反例，若有以下三个活动：(1,5)，(5,9)，(4,6)。该三个活动按此策略应选第三个，但第三个活动与其余活动冲突，故也不是最合适的方法。

- 早结束的活动优先？-------√

本题目的是要安排尽可能多的活动，显然，早结束有可能再安排其他活动。

由此确定贪心策略：优先选取结束时间早且与已选择的活动相容的活动作为相容集合中的元素，以便为未安排的活动留下尽可能多的时间。也就是说，该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动。

考虑以下示例数据(数组 B 存放活动开始时间，E 存放结束时间，已按结束时间排序)：

i	1	2	3	4	5	6	7	8	9
$B[i]$	1	2	0	5	4	5	7	9	11
$E[i]$	3	5	5	7	9	9	10	12	15

利用贪心策略，可以对表中的活动作以下选择：

① 选取活动 1 放入相容集合 A，A：{1}，因为其结束时间最早。

② 在与活动 1 相容的待定集合{4,5,6,7,8,9}中,选择结束时间最早的活动 4 放入相容集合 A,A:{1,4}。

③ 在与活动 1 和 4 都相容的待定集合{7,8,9}中,选择结束时间最早的活动 7 放入相容集合 A,A:{1,4,7}。

④ 在待定集合中选择与活动 1、4、7 都相容的活动 9 放入相容集合 A,得到最终的相容活动安排{1,4,7,9}。

为了帮助读者更好地理解以上过程,可以用下面的时间分布如图 2-3-5 来直观地描述整个活动的安排过程。

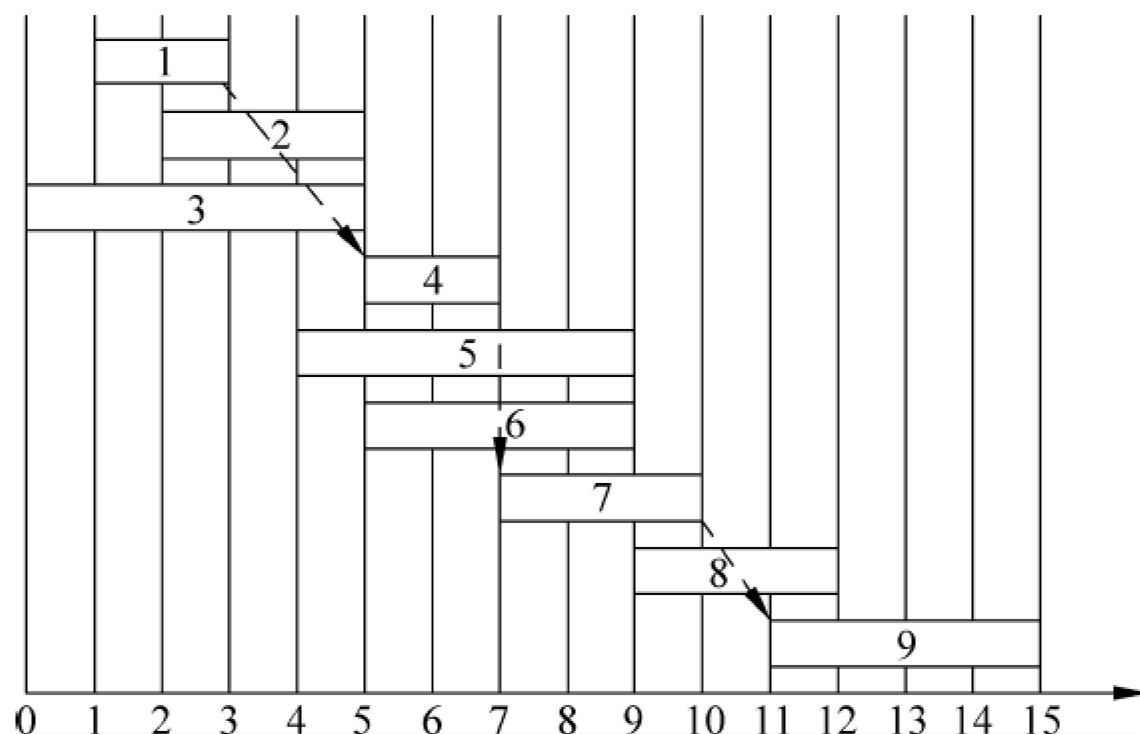


图 2-3-5　用时间分布图描述活动安排

算法(伪代码形式)如下:

① 对所有活动按结束时间的先后排序(早结束者在前)。

② 记录第一个活动,并使 $j=1$。

③ 循环(i 从 2 到 n)。

若第 i 个活动的开始时间≥第 j 个活动的结束时间:

记录该第 i 个活动

$j=i$

④ 输出所有记录的活动。

3. 贪心算法的适用性

贪心算法要想找到最优解,需要两个条件。

第一,原问题具有最优子结构,这样才能得到局部最优解。

第二,原问题的最优解包含了它的子问题的最优解。这样才能由局部最优合成全局最优,并且局部最优解一旦获得就不会改变。

有时候,选择局部最优解并不能得到全局最优解。例如,在前面售货员找钱的例子中,如果现有货币体系中还有 8 元面值且不存在 2 元面值的话,则应该找零给顾客(65.5 元)50 元一张、8 元一张、5 元一张、1 元两张和 5 角一张,共有六张零钞。但若给 50 元一张、5 元三张和 5 角一张,显然只需五张零钞。

【例 2-3-10】 石子合并问题。

在操场的四周摆放 N 堆石子,每堆石子有数值大小(表示数量等权重)。现要将石子有次序地合并成一堆。规定每次只能选相邻的两堆合并成新的一堆,并将新的一堆石子数记为该次合并的得分。

已知每堆石子数量,请选择一种合并石子的方案,使得进行 $N-1$ 次合并得分的总和最小。

分析:能否使用贪心策略?

对于这个问题,很多人都会选择贪心算法求解——每次选相邻之和最小的两堆石子合并。例如,对于图 2-3-6,利用此贪心策略的确可以找到最优解。

最小值为(4+4)+(8+5)+(13+9)=43。

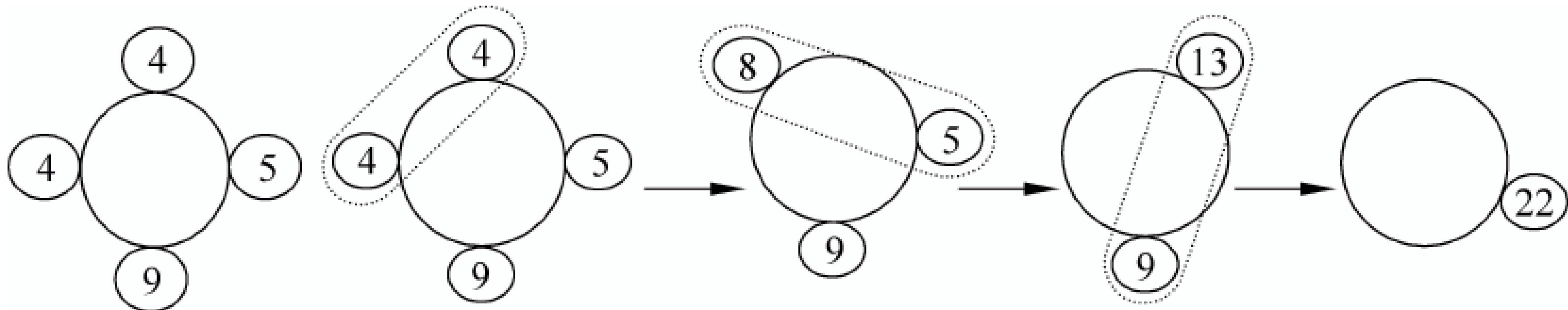


图 2-3-6 使用贪心策略求解石子合并问题(正例)

然而本题给的样例数据实际上是一个"陷阱",造成了用贪心法即可解决的假象。现在看一个反例,如图 2-3-7 所示。

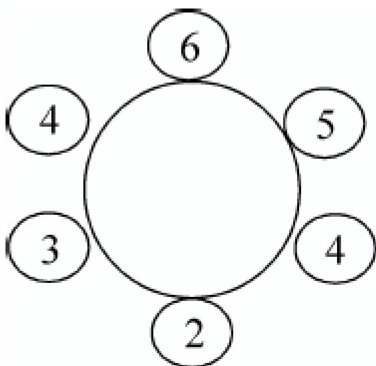


图 2-3-7 石子合并(反例)

- 根据贪心算法的合并方案,其逐次合并得分为 2+3=5,4+5=9,4+5=9,9+6=15,15+9=24

于是得分总和为 5+9+9+15+24=62。

- 另一种合并方案的逐次合并得分为 2+4=6,3+4=7,5+6=11,7+6=13,11+13=24

于是得分总和为 6+7+11+13+24=61。

可见此问题用贪心算法得到的不是最优解,贪心算法在子过程中得出的解只是局部最优,而不能保证使得全局的值最优,需要用其他算法(如动态规划)来解决。限于篇幅,这里不深入讨论。

2.4 本 章 小 结

本章介绍了计算机程序设计中算法的概念和几种常用的算法。由于涉及顺序、选择、循环三种基本的程序控制结构,枚举、迭代、递推和贪心算法可放在第 4 章后面讲解。

本章要点如下:

(1) 算法是问题的程序化解决方案。要使计算机能完成人们预定的工作,首先必须为如何完成该工作设计算法,然后再根据算法编写程序。

(2) 算法具有正确性、可行性、确定性和有限性。此外,一个合格的算法,对于输入数据应具备健壮性,输入输出应具备良好的人机交互界面。

(3) 可以使用自然语言描述算法,但不够简练、直观和清晰。为此,人们尝试所谓"伪代码"——用简练的语言和符号,以类似程序的格式描述算法。更有效的方法是用图形来描述算法,常用的是流程图。本章介绍了算法流程图的基本图形构件,以及通过面向流程图用户的专业网站 ProcessOn 绘制流程图的方法。

(4) 比较详细地介绍了枚举算法、迭代和递推算法、贪心算法的基本思想、求解问题策

略和步骤,这些对程序设计初学者是非常重要的,希望读者一定要静下心仔细阅读,细心领会,对今后的程序设计大有益处。

2.5 习题与思考

2-1 选择题。

(1) 算法应具有正确性、可行性、无二义性和________。

A. 简单性　　B. 有限性　　C. 结构性　　D. 可运行性

(2) 下列________是对算法的正确描述。

A. 解决一个问题只有一种算法

B. 对于所有问题都可以找到最好的算法

C. 算法所包含的语句数量越少,算法越先进

D. 解决一个问题可以有多种算法

(3) 评判一个算法的优劣,可以________。

A. 只考虑能否得出正确的答案

B. 只考虑算法执行的时间

C. 只考虑算法所需占用的空间

D. 从算法执行时间和需占用的空间两方面考虑

(4) 以下________不是流程图常用的图框。

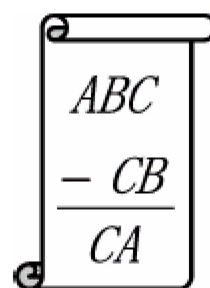
A. 矩形框　　B. 平行四边形框　　C. 三角形框　　D. 菱形框

(5) 把求解问题分成若干个子问题。对每一子问题求得局部最优解,最后得到原来问题的解,这是________算法的基本思想。

A. 贪心　　B. 分治　　C. 枚举　　D. 递推

2-2 写出求解右边数学灯谜的枚举算法。

求要使算式成立对应的 A、B、C 值(A、B、C 均为一位正整数)。



2-3 某人有四张 3 角的邮票和三张 5 角的邮票。请用枚举法写一算法(伪代码形式),求用这些邮票中的一张或若干张可得到的所有不同邮资。

【提示:使用两轮循环,邮资$=3i+5j$,i 为 3 角邮票枚举数(0～4)、j 为 5 角邮票枚举数(0～3)。】

2-4 一个顽猴在一座有 30 级台阶的小山上跳跃爬山,一步可以跳 1 级或 3 级台阶。使用递推算法求该顽猴上这 30 级台阶共有多少种不同的爬法。

【提示:第一级顽猴只能跳 1 步,即一种跳法;第二级也只能一步一步跳,也是一种跳法;第三级可以一步一步跳三次,或者直接一次跳三级台阶,所以有两种跳法,即初始时 $f(1)=1$,$f(2)=1$,$f(3)=2$。顽猴最后一步到顶时,或者位于第 29 级台阶(跳一步),或者位于第 27 级台阶(跳三步),即 $f(30)=f(29)+f(27)$,以此类推可得递推关系:$f(k)=f(k-1)+f(k-3)$,$k>3$。】

2-5 两个人分 n 个大饼,一次可以拿 1 个或者 2 个,不吃完不允许再拿。假定两人吃饼的速度相同,试画出先拿者采用贪心算法的流程图。其中 n 为开始输入的大饼总数。

每次后拿者拿饼的数目(1 个或 2 个)也为即时输入。算法最后输出先拿者所拿的大饼总数。

【讨论:此问题先拿者采用贪心算法(即每次都尽量拿 2 个)是否一定能得到最多?

实例一:两个人分 5 个大饼,A 用贪心方法:第一次拿 2 个,B 拿 1 个,那么当 B 吃完 1 个后(A 的 2 个还没有吃完)还可以拿 2 个,最终拿了 3 个。A 只拿了 2 个。所以 A 用贪心策略不能得到最多。

实例二:两个人分 7 个大饼,A 用贪心方法:第一次拿 2 个,B 若拿 1 个,那么当 B 吃完 1 个后还可以拿 2 个,共拿了 3 个。但 A 吃完 2 个后又可以拿 2 个,最终为 4 个;若 A 第一次拿 2 个后,B 也拿 2 个,那么当 A 先吃完后还可以拿 2 个,共 4 个。但 B 吃完 2 个后只剩下 1 个了,最终也只拿 3 个。所以这里 A 用贪心策略可得最多。】

2.6 实验 算法描述和绘制流程图

2.6.1 实验目标

(1) 掌握流程图的基本制作方法。

(2) 掌握使用 ProcessOn 软件绘制流程图的操作。

(3) 熟悉了解枚举算法、迭代和递推算法、贪心算法。

2.6.2 实验范例

1. 根据 2.2.3 节介绍的方法,使用 ProcessOn 软件绘制本章例 2-3-2 流程图

① 打开浏览器,输入网址:http://www.processon.com,进入 ProcessOn 网站,如图 2-2-3 所示。如果没有注册,按要求进行注册;如果已经注册,单击“马上体验”按钮,然后输入本人邮箱地址和注册密码后进入。

② 在“文件名”对话框中输入欲绘制流程图保存的文件名(例如:例 2-3-2),单击“确定”按钮进入绘图界面。

③ 在左边的图形工具库中选择“Flowchart 流程图”,如图 2-6-1 所示,显示流程图工具集。

④ 从流程图工具集中拖曳“开始/结束”图框到右侧画布上,然后在图框中输入文字“开始”。

⑤ 将光标移到所画图框上,当变为十字形光标时,往下拖曳图框下方的控制点,绘出流程线。松开鼠标后,画布上自动弹出流程图图形列表,如图 2-6-2 所示。直接单击一个需要的图形即自动加入画布并连在所画流程线箭头下。

⑥ 按相似的操作画出例 2-3-2 整个流程图图形。在绘制过程中,可以直接拖曳图框移动位置,可拖曳图框角点改变大小,也可以拖曳流程线改变长短和位置。通过菜单下面的工具栏可设置文本格式,双击流程线后也可在线上标注字符和文字,如图 2-6-3 所示。单击图框或流程线后,按 Delete 键可将其删除。

⑦ 流程图绘制后已自动保存在网站本人文件夹下。通过“文件”菜单中的“下载为”,可下载到本地计算机中,保存为 PNG 图形或 PDF 文件。



图 2-6-1　显示 Flowchart 流程图

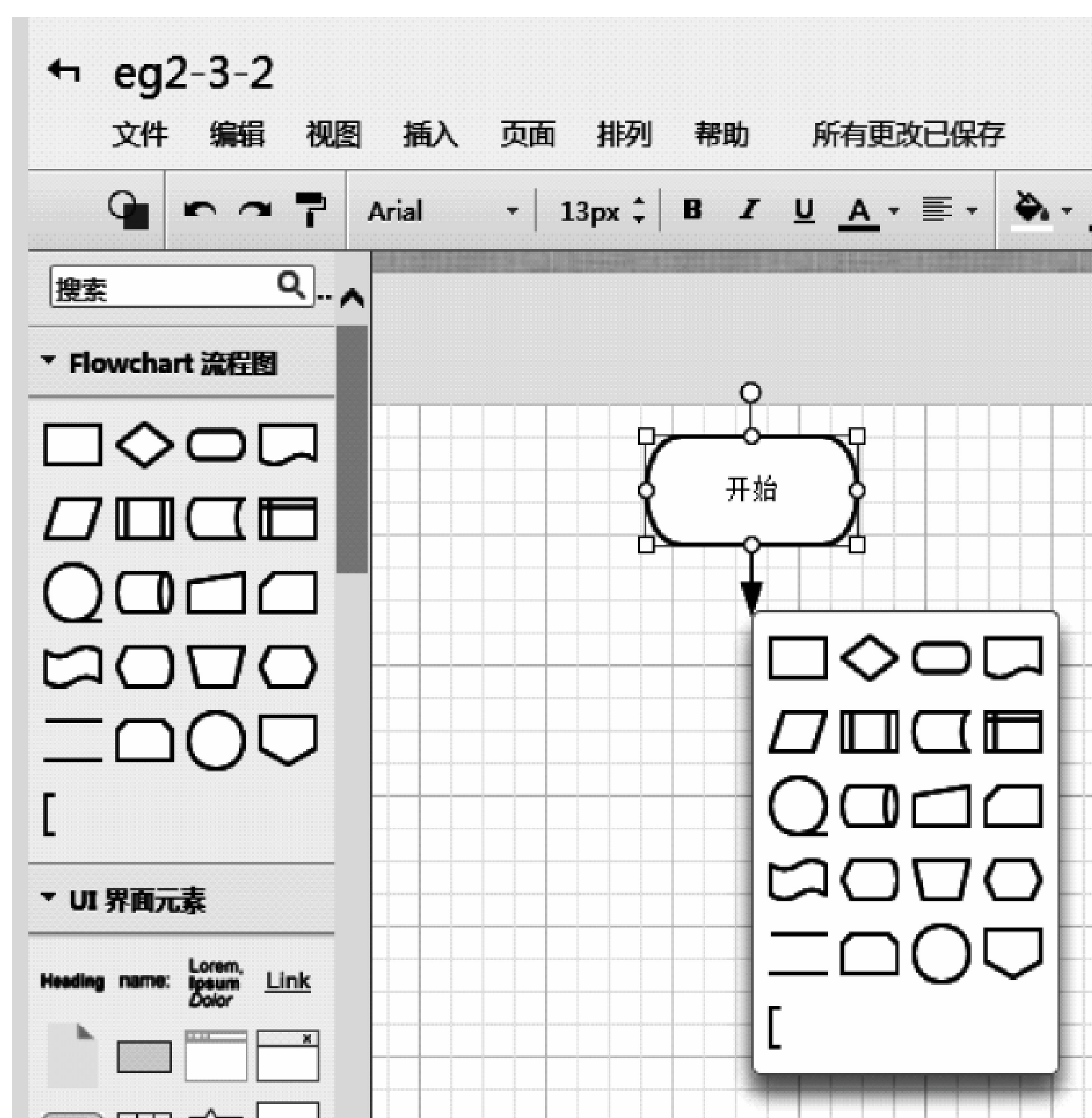


图 2-6-2　选择下一个图形

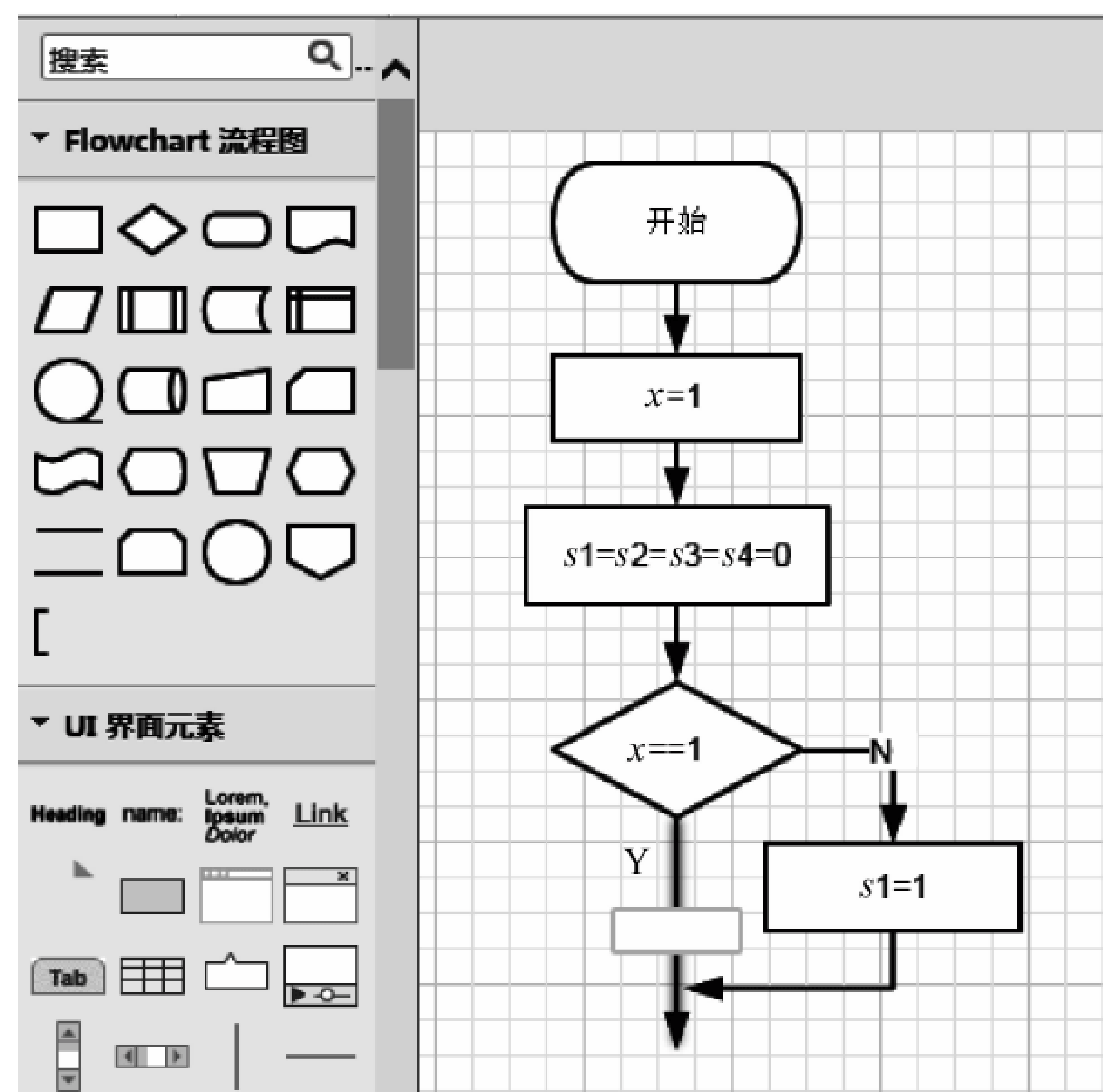


图 2-6-3　在流程线上标注字符

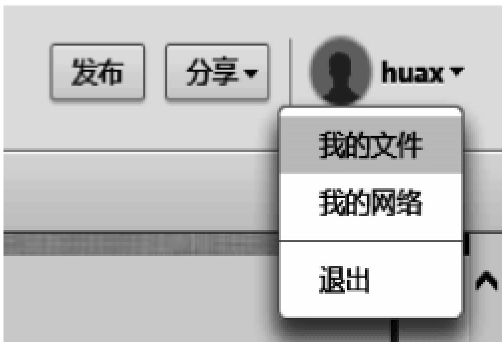


图 2-6-4　进入本人文件夹

⑧ 鼠标单击右侧用户名下拉列表中的“我的文件”，如图 2-6-4 所示，进入“我的文件”窗口界面，如图 2-6-5 所示。在该窗口中可进行本人文件和文件夹的操作维护。

2. 求两个数的最大公约数算法

(1) 问题描述：输入两个正整数，求它们的最大公约数。

(2) 算法设计。

算法 1. 直接用最大公约数的定义。



图 2-6-5　“我的文件”窗口

伪代码形式描述如下：

① 输入两个正整数。
② 如果 $a<b$，交换 a、b 值。
③ 循环（i 取值从 b 到 1）。
　如果 a 被 i 整除，
　　i 为最大公约数，
　结束循环。

要求：将该算法用流程图表示。

算法 2. 使用“辗转相除”算法。

原理：对两个数 p 和 q，求它们的最大公约数（greatest common divisor）。

gcd(p，q)＝gcd(q，p％q)，当 p％q 为 0 时，两数的最大公约数即为 q。

即：将大数作为被除数，小数作为除数，若二者余数不为 0，则将小数作为被除数，余数作为除数，…直到余数为 0。

例如，求 252 和 105 的最大公约数，计算过程如下所示：

252％105，余数为 42；

105％42，余数为 21；

42％21，余数为 0；

由此可得，252 和 105 的最大公约数为 21。

要求：据此算法绘制流程图（根据图 2-6-6 给出的图形绘制并补全空白部分）。

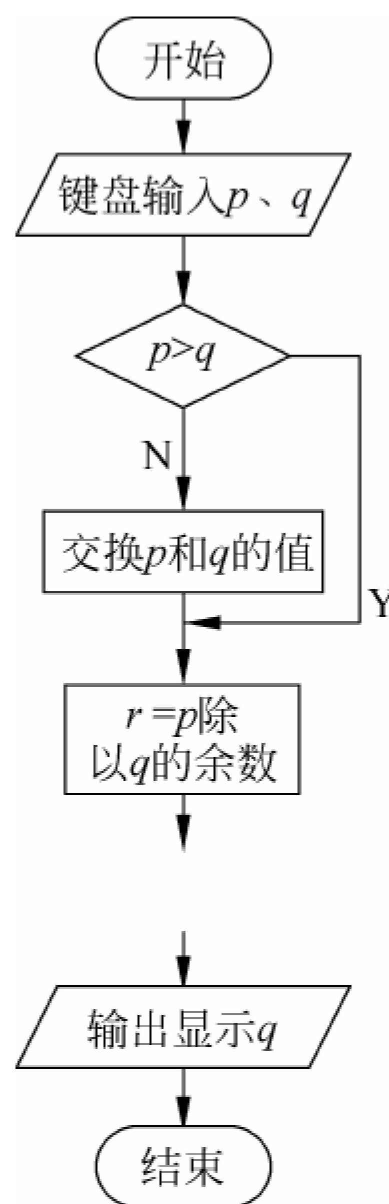


图 2-6-6　完成求最大公约数流程图

2.6.3　实验内容

（1）流程图基本结构训练。

① 使用流程图设计一个算法：接收一个输入的整数，输出其个位和十位的数字。

② 使用流程图设计一个分支算法：接收一个输入的数，如果是一个正整数，显示“有效数！”，如果是一个 1～100 的正整数，显示“合理数！”，否则，显示“无效数！”。

③ 使用流程图设计一个循环算法：计算 1＋3＋5＋…＋99。

（2）画流程图训练。绘制图 2-3-2 验证谷角猜想的算法流程图。

（3）根据例 2-3-8 最优装载问题给出的伪代码形式算法，做出对应的流程图。

（4）写一个算法（流程图形式）：输入三个数，输出其最大者。

（5）写一个算法（伪代码或流程图）：输入三根钢管的长度（如 240cm、200cm、480cm），计算把它们截成同样长度的小段（不得有短节损耗），每段最长可以是多少？

第3章 数据表示和计算

计算机的程序处理数据生成有用的信息，写程序就是描述数据的处理过程，其中必然涉及数据的描述和从数据出发的计算。编程语言提供符号化的手段表示现实世界中的各种信息。例如：求三门课程成绩的平均分，四舍五入到个位数，在 Python 中可以写出下面加粗部分表示的“表达式”。

```
>>> import math
>>> math.trunc((98 + 95.5 + 85)/3 + 0.5)
93
```

加粗部分的表达式中包含了一些数据，如整数和实数等，还包含了对数值数据的计算即加法(+)和除法(/)，以及实现取整的数学函数 trunc。

要理解和正确地写出这个表达式，必须要知道 Python 语言对各种基本数据在写法上的规定，还需要了解表达式如何表示各种基本数据的运算，以及计算的结果是什么，这些是本章要讨论的主要问题。

在理解了基本数据之后，再讨论如何在基本数据的基础上去组织文本数据和批量数据，以表示和处理更复杂的复合数据。

3.1 数据和数据类型的概念

3.1.1 数据的表示

在计算机的世界中，数据是对现实世界的抽象。使用计算机解决现实世界中的问题时，首先要分析有哪些信息是解决问题所需要的，并将它们表示成计算机能够接受的形式。所以使用计算机解决问题的第一步是挖掘具体数据对象与所解决的问题相关的信息，加以描述和表示，而忽略那些与所解决的问题无关的信息，这个过程就是抽象，抽象可以梳理计算问题所关注的主要数据并加以表示。程序中的数据一般会包括数值数据、文本数据和复合类数据。

假设为了预测 PM2.5 浓度的走势，需要记录所测得的 PM2.5 浓度值，一个测得的 PM2.5 浓度值为 60μg/m³，在 Python 语言中可以用整数 60 来表示，也可以用浮点数 60.0 来表示。这一类数据都会以数值数据来表示，以方便对 PM2.5 的浓度值进行数学计算。

又如，在一个英语学习的程序中，表示一个英语句子的方法可以是双引号括起来的字符串"my English words "，也可以用单引号括起来'my English words '。这一类数据以文本数据来表示，可以对文本数据进行大小写转换、取子串等文本操作。

又如,在一个学籍管理的计算机信息系统中,需要表示现实中的学生,不可能把所有的学生信息都录入系统中,如学生头发的长短、性格、着衣风格等信息就不是学籍管理所要关注的,学籍管理关注的是学生的学号、姓名、性别、出生日期、入学日期、专业等,学生的数据类型会抽象为(学号,姓名,性别,出生日期,入学日期,专业编码),对应一个具体的学生王小明,他的数据为(10132110115,'王小明','男',1993-2-18,2013-9-1,21601),这就完成了从现实世界的王小明到计算机世界的王小明的抽象,如图 3-1-1 所示。

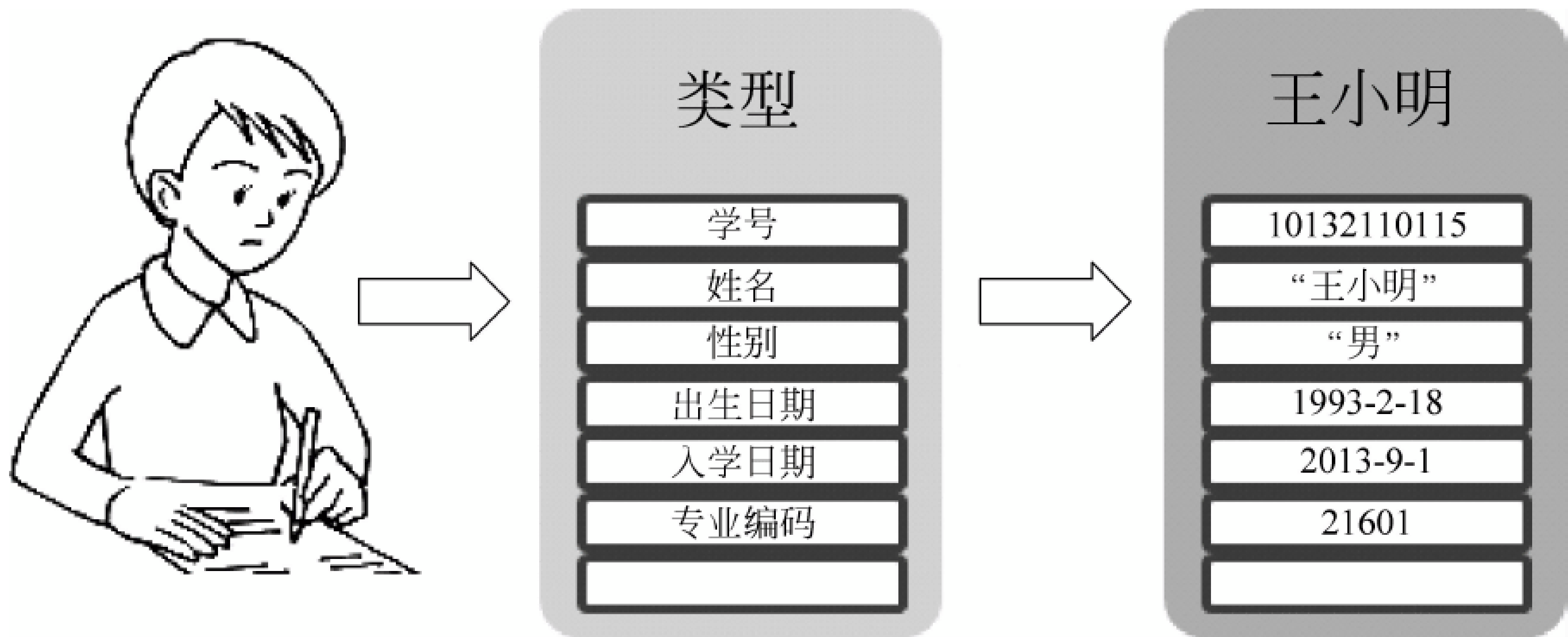


图 3-1-1　数据的抽象

3.1.2　数据类型的概念

1. 数据与数据类型的关系

计算机的世界是二进制的世界,0 和 1 描述着计算机世界中的指令、地址、数据等。数据是未经组织的要素,进入计算机后由程序处理得到相应的信息,计算机再以不同的形式展示这些信息(文字显示、多媒体呈现、打印、存储文件等)。数据进入计算机并交由程序处理之前,先要存储在内存中,所以程序数据的表示与内存的组织方式密切相关。

内存单元的管理是以字节(8b,一位 0 或 1 称为 1b)为单位进行分配和回收的。在内存中存储和处理数据与现实世界不同,是区分数据类型的,不同的数据类型占用不同的字节数,并有着不同的编码和运算机制。例如一个整数 20 要存入内存,整数在大多数计算环境中占 4 个字节,整数在硬件中的编码直接转化为二进制,所以在内存中存储的形式为:0000 0000 0000 0000 0001 0100,而不是 1 0100。可以这么说,数据类型决定程序数据的表示。

2. 数据类型的定义

数据类型是一组数据及在这组数据上的运算,它包含三方面的含义:

一是存储结构,一种数据类型的数据由几个字节构成,存储的空间大小确定,它可以表示的数据范围也就确定了。也就是说,计算机数据表示是受限制的,没有数学中“无限”的概念。

二是存储机制,即如何解释比特流,各种数据类型的编码方式是怎样的。

三是运算,即每种数据类型可以执行的运算有哪些。每一种数据类型有着不同的运算集,也就是对不同数据类型的数据的操作是不同的。同一运算符号,不同的数据类型也有着不同的解释。例如“+”,整数类型的解释是加法,3+5 得到 8,但文字类型的解释是连接,3+5 得到 35。

综上所述,当数据具有以下相同的特性时就构成一类数据类型:

- 采用相同的书写形式。
- 在具体实现中采用同样的编码形式(内部的二进制编码)。
- 在内存存储中具有相同的二进制编码位数。
- 能做同样的运算操作。

一般说来,学习计算机解决实际问题要从学习数据类型入手,了解某一种编程语言提供了哪些数据类型支持对数据的表示。学习每一种数据类型时,要学习这种数据类型的 4 个特征,了解数据类型能表示怎样的数据,对这些数据能做怎样的操作。

3.1.3 Python 的内置类型

每一种编程语言都会预先设置一些基本的数据类型,称为内置类型,并允许在内置类型的基础上构造更复杂的数据类型。

Python 的内置类型如图 3-1-2 所示,主要区分为简单类型和容器类型,简单类型主要是数值型数据,包括整型数据、浮点型数据、布尔类型数据和其他语言不多见的复数数据。容器类型可以应用于一次处理多个对象的场合,包括字符串 str、元组 tuple、列表 list、集合类型 set、字典类型 dict。

简单数据类型	序列对象	其他类型
•整型 int •浮点型 float •复数 complex •布尔类型 bool	•字符串 str •元组 tuple •列表 list	•集合类型 set •字典类型 dict

图 3-1-2　Python 的主要数据类型一览表

其中支持访问给定顺序的对象的称为序列对象,包括字符串 str、元组 tuple、列表 list。字符串,可方便地表示文本数据,元组和列表可以表示数据的序列。除序列对象之外的容器对象包括集合和字典。集合也可以表示数据的组合,但是其中的数据的存储不是连续有序的,与序列类型的区别在于不能按下标索引。字典是 Python 中唯一内置映射数据类型,字典元素由键(key)和值构成,可以通过指定的键从字典访问值。

3.1.4 常量和变量

1. 常量和变量

通常,程序中数据有两种表示方式:常量和变量。

按接近人的习惯设计并加以不同数据类型的区别,称为数据的文字量,是数据的“书写形式”。例如整数 389,浮点数 23.56,字符串 hello,这些数据是不会改变的,也称为字面常量;在本章的后继小节对数据类型的具体介绍中将讲述每一种数据类型的常量的书写形式。

变量描述的是存储空间的概念,将数据存储在内存中,内存空间就是可操作的变量,用一个名称来引用内存空间,这个名称称为变量名。变量的值是可以变化的。

【例 3-1-1】 将 3.141 592 6 存储在变量 a 中,显示 a 的值为 3.141 592 6,当再次将

3.1415 存储到变量 *a* 中，显示 *a* 的值为 3.1415。

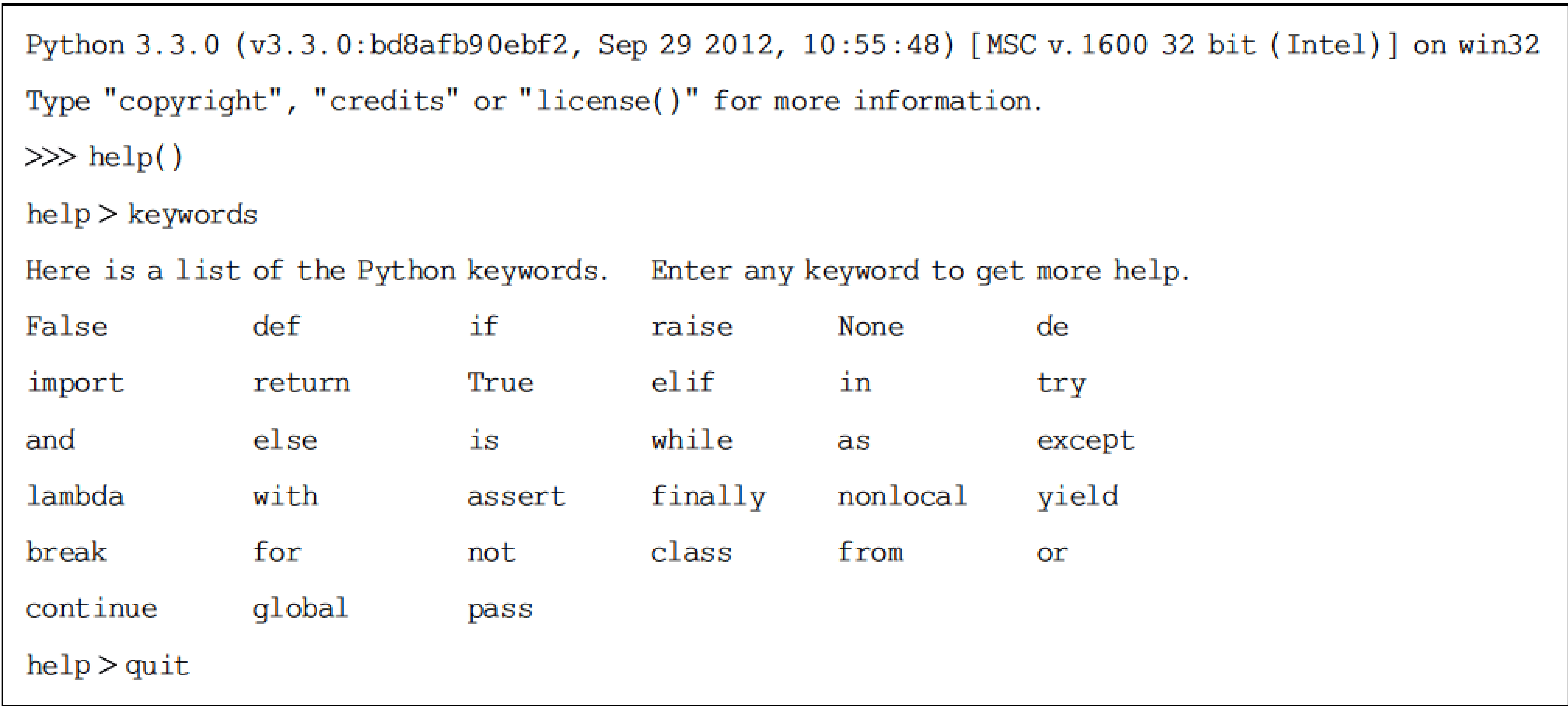
```
>>> a = 3.1415926
>>> a
3.1415926
>>> a = 3.1415
>>> a
3.1415
>>>
```

2. 标识符和变量名

标识符是指在程序书写中程序员为一些特定对象的命名，包括变量名、函数名、类名、对象名等。Python 中的标识符由大小写英文字母、数字、下划线组成，以英文字母、下划线为首字符，长度任意，大小写敏感。

为了增加程序的可读性，通常使用有一定意义的标识符命名变量，但标识符不能与 Python 关键字同名。

Python 的关键字随版本不同有一些差异，可以使用 help 函数查阅，Python 3.3 版的关键字如图 3-1-3 所示。



```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> help()
help> keywords
Here is a list of the Python keywords.  Enter any keyword to get more help.
False       def        if        raise      None        de
import      return     True      elif       in          try
and         else       is        while      as          except
lambda      with       assert    finally    nonlocal    yield
break       for        not       class      from        or
continue    global     pass
help> quit
```

图 3-1-3　Python 关键字

3. 变量的基本操作

1) 变量的赋值

例 3-1-1 中出现的＝不是数学中的等号，在程序语言中称为赋值运算符，赋值语句的语法格式为

<变量> = <表达式>

赋值运算符的定义是将右边表达式的值赋给左边的变量，即将数据存储到变量所引用的内存空间中。

【例 3-1-2】 通过下面一组操作来理解变量的操作。

```
>>> x = 10
>>> y = 10 * x
```

```
>>> x = x + y
>>> x
110
>>> y
100
```

第一个表达式 $x=10$：先将整数常量 10 赋值给变量 x，赋值后变量 x 引用的内存存储单元存放了整数 10，第二个表达式 $y=10*x$：先从变量 x 中读取整数值 10，再参加乘法运算得到 100，将整数 100 赋值给变量 y。第三个表达式 $x=x+y$：先从变量 x 中读取整数值 10，再从变量 y 中读取整数值 100，再参加加法运算得到 110，最后将整数 110 赋值给变量 x。

赋值运算不难理解，但很容易与数学中的等号相混淆，例如在数学中下面的等式是正确的：

$$Y+2=X$$

读作：Y 加 2 等于 X。但根据赋值运算符的定义，在程序中显然是个错误的表达式，赋值运算符的左边必须是一个变量，用于接收右边表达式的值，不是等值的概念。这也就能够理解下面的式子：

$$X=X+2$$

在程序中解释为先读取 X 的值，再参加加法运算，加 2 后的值，再重新赋值给 X。这样的一类计算过程中，执行对变量自身增值的操作，该变量称为累加器。

程序中使用＝为变量赋值，注意与关系运算中的等于(＝＝)区分。等号左边是变量名，右边可以是常量或表达式。例如以下都是有效的赋值语句：

X=10　yϕ=0.5　_yes_no=True　number=52.5+x　Boy='boy'

在 Python 中，与许多编程语言不同，还允许同时对多个变量赋值，即所谓的“并行赋值”。

【例 3-1-3】 变量的并行计算。

```
>>> x,y,z = 3,4,5
>>> x
3
>>> y
4
>>> z
5
```

通过并行赋值能直接通过赋值语句交换两个变量的值，而不需借助中间变量。

【例 3-1-4】 交换两个变量的值。

```
>>> x,y = y,x
>>> x
4
>>> y
3
```

对于常用的数学运算符(＋、－、*、/、** 等)，还可以使用增强的赋值运算符形式。例如 $x=x+1$，也可写成 $x+=1$ 的形式，称为复合赋值运算。这种形式不仅表达更精炼，由于

在运行时仅需查询一次变量的值，因而执行速度也较快。

【例 3-1-5】 复合赋值运算符示例。

```
>>> x,y,z = 3,4,5
>>> x += 1
>>> y * = 2
>>> z ** = 3
>>> x,y,z
(4, 8, 125)
```

2）变量的引用

使用变量的方式就是将变量直接写在表达式中。计算过程中遇到变量，就会取变量的值参加计算。

在表达式中可以反复读取一个变量的值。

【例 3-1-6】 求解三边为 3cm，4cm，5cm 的三角形面积的表达式，其中求平方根的函数为 sqrt。

没有变量参与时可写出：

```
>>> import math
>>> math.sqrt( ((3 + 4 + 5)/2) * ( ((3 + 4 + 5)/2) - 3) * (((3 + 4 + 5)/2) - 4) * (((3 + 4 + 5)/
2) - 5) )
6.0
```

增加中间变量 s，表达式简短清楚多了。

```
>>> s = (3 + 4 + 5)/2
>>> math.sqrt( s * ( s - 3) * (s - 4) * (s - 5) )
6.0
```

s 通过计算获取了三条边之和的一半的值后，在计算面积的公式中可以反复读取。

由于变量的值是可以变化的，表达式依赖变量的值就可以计算出不同的结果，与只包括常量数据的表达式有本质的不同。程序要解决的问题应有代表性，能够解决一类问题。

如果仅仅如上面求三边为 3cm，4cm，5cm 的三角形的面积，只需要一个计算器就能解决问题，编写一个程序，通常要解决的是求任意一个三角形的面积，这就存在一个数据建模的过程。

这个程序要解决的问题可描述为已知三角形的三条边，求三角形的面积。问题解决的方法这里采用的是通过三角形的面积公式求解。在求三角形的面积公式中，a、b、c 表示三条边，s 表示三条边之和的一半，area 表示三角形的面积。当 a、b、c 赋予不同的值时，就可以求解不同的三角形的面积。a、b、c、s、area 就是程序中解决问题所需的数据模型。

【例 3-1-7】 对求解三角形的过程再做以下修改，当对 a，b，c 赋予不同的值时，可以计算得到不同三角形的面积。

```
>>> import math
>>> a,b,c = 3,4,5
>>> s = (a + b + c)/2
>>> area = math.sqrt( s * ( s - a) * (s - b) * (s - c) )
```

```
>>> area
6.0
>>> a,b,c = 12,33,25
>>> s = (a + b + c)/2
>>> area = math.sqrt( s * ( s - a) * (s - b) * (s - c) )
>>> area
126.8857754044952
```

试想如果把这段代码写到一个程序,a、b、c 的取值可以在程序运行时从键盘输入,那么这个程序可以处理计算任意一个三角形的面积。

3.1.5 Python 的动态类型

从计算机硬件的角度考虑,数据是存储在内存的存储单元中的,CPU 获取存储地址,访问内存的存储单元。

从程序的角度考虑,变量所引用的数据空间可视为一个容器,对应内存的存储单元,程序运行时,可以把数据存入变量所引用的数据空间,也可以读取变量所引用的数据空间的值,每一个变量都有一个名字,程序中按名字访问变量,进行存取工作。

编译器在将高级语言翻译为机器语言时,将变量名对应到内存地址。

许多程序设计语言需要预先指定变量的数据类型,变量是区分不同数据类型的,不同类型的变量中存储特定数据类型的数据,数据类型确定了,一个变量对应内存的字节数,对应的编码形式和可参加的运算也就确定了。一旦存入的数据与预先声明的数据类型不一致,程序就会出错,这种程序设计语言称为静态类型化的语言。

Python 语言使用“动态类型”技术,变量使用前不需要声明数据类型即可使用,然后根据变量存放的数据不同,决定其数据类型。通过 type(<变量>)函数可以检测变量的数据类型。Python 程序在创建每一个数据对象时,给每一个数据对象设置一个对象 ID。使用函数 id(<变量>),可以得到对象的 ID。

【例 3-1-8】 动态类型示例。

当对 x 赋值整数 354 时,Python 在内存中创建整数对象 354,并使变量 x 指向这个数据对象,x 的类型为整型 int,此时 x 所指向的对象的 ID 为 34 539 888。

```
>>> x = 354
>>> type(x)
<class 'int'>
>>> id(x)
34539888
```

再次对 x 赋值“word”时,Python 在内存中创建字符串对象“word”,并使变量 x 指向这个字符串数据对象,x 的类型变为字符串 str,x 所指向的对象的 ID 为 33 407 296,参见图 3-1-4。

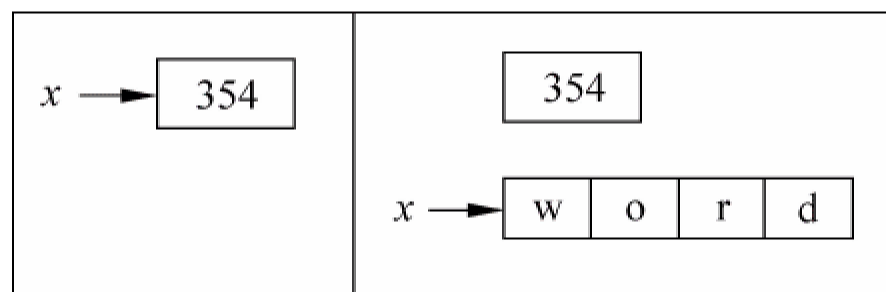


图 3-1-4 Python 的动态类型技术

```
>>> x = "word"
>>> type(x)
<class 'str'>
```

```
>>> id(x)
33407296
```

也就是说,并不是 x 所代表的内存空间的内容发生了改变,而是 x 指向了存储在其他内存空间的另一个对象。当 x 从整数对象 354 转向字符串对象 word 后,整数对象 354 没有变量引用它,它就成了某种意义上的"垃圾",Python 会启动"垃圾回收"机制回收垃圾数据的内存单元,供其他数据使用。

3.2 数值数据的表示与计算

3.2.1 数值数据的常量表示

Python 的数值数据包括整型数据、浮点型数据、布尔类型数据和复数数据。

1. 整型数据 int

Python 的整数大小只受机器内存大小的限制,默认情况下采用十进制,但也可采用其他进制形式。

例如:0o137(八进制的 95,o 表示八进制数)、0b1111(二进制的 7,b 表示二进制数)、0xff(十六进制数的 255,x 表示十六进制数),使用 type 函数可以获取数据的类型。

【例 3-2-1】 int 数据示例。

```
>>> 0o137
95
>>> 0b111
7
>>> 0xff
255
>>> type(28346283742874)
<class 'int'>
>>> type(0o137)
<class 'int'>
```

2. 浮点型数据 float

浮点型数据支持小数形式表示和指数形式表示。

例如 12 是整数,12.0 就是浮点数,8.9e−4 表示 8.9×10^{-4} 即 0.000 89。计算机中的浮点数都是以近似值存储数据的,Python 的 float 类型数通常可提供至多 17 个数字的精度,例如:print(23/1.05)显示的值为 21.904 761 904 761 905。

【例 3-2-2】 float 数据示例。

```
>>> type(12)
<class 'int'>
>>> type(12.0)
<class 'float'>
>>> 8.9e-4
0.00089
>>> type(1.2e1)
<class 'float'>
```

```
>>> 23/1.05
21.904761904761905
```

3. 布尔类型数据 bool

Python 的布尔类型数据只有两个：True 和 False，表示真和假。注意书写时要区分大小写。以真和假为值的表达式称为布尔表达式，用于表示某种条件是否成立，以支持选择控制和循环控制中必不可少的条件判断。

【例 3-2-3】 布尔数据示例。

```
>>> type(True)
<class 'bool'>
>>> x,y = 10,20
>>> x > y
False
>>> x + 10 <= y
True
```

4. 复数类型数据 complex

Python 提供的复数类型数据也是它的一大特色。复数由实数部分和虚数部分构成，表示为

```
real + imag(J/j 后缀)
```

实数部分和虚数部分都是浮点数。

【例 3-2-4】 复数的常用操作示例。

```
>>> aComplex = 4.23 + 8.5j
>>> aComplex
(4.2300000000000004 + 8.5j)
>>> aComplex.real  # num.real 返回复数的实数部分
4.2300000000000004
>>> aComplex.imag  # num.imag 返回复数的虚数部分
8.5
>>> aComplex.conjugate()  # num..conjugate() 返回复数的共轭复数
(4.2300000000000004 - 8.5j)
```

3.2.2 数值数据的计算

1. 表达式

表达式是数据对象和运算符按照一定的规则写出的式子，描述计算过程。例如算术表达式由计算对象、算术运算符及圆括号构成。最简单的表达式可以是一个常量或一个变量。

【例 3-2-5】 请列出计算半径为 4.5 的球的体积的表达式，math.pi 表示 π。

```
>>> 4 * (math.pi * 4.5 * 4.5 * 4.5)/3
381.7035074111598
```

数值数据可参与的运算包括算术运算、关系运算、逻辑运算，赋值运算，如表 3-2-1 所示。

表 3-2-1　数值对象的运算符

运　算　符	描　　述
$x+y$,$x-y$	加、减
$x*y$, x/y,$x//y$, $x\%y$,$x**y$	相乘、相除、整除、求余、求乘方
<,<=,>,>=,==,!=	比较运算符
or,and,not	逻辑运算符
=,+=,-=,*=,/=,%=,**=	赋值运算,复合赋值运算符

2. 数值数据的运算

1）算术运算

Python 提供的算术运算包括加、减、乘、除和求与运算,与数学中的算术运算的定义基本相同,不同的地方有 Python 支持的除法区分为普通的除法和整除。

【例 3-2-6】 整数的除法和整除运算示例。

```
>>> x = 8
>>> y = 3
>>> x/y
2.6666666666666665
>>> x//y
2
```

【例 3-2-7】 浮点数的除法和整除运算示例。

```
>>> x = 3.8
>>> y = 0.7
>>> x/y
5.428571428571429
>>> x//y
5.0
```

% 为求余数的运算,可以通过求余运算来判断一个数是否能被另一个数整除

【例 3-2-8】 判断一个数是否是偶数。

```
>>> x = 834
>>> x % 2 == 0
True
```

2）关系运算

数值运算的关系表达式由数值数据和关系运算构成,得到的结果为布尔类型数据:True 或 False,一般形式为

<数值 1><关系运算符><数值 2>

关系运算符包括<、<=、>、>=、==、!=,分别表示小于、小于等于、大于、大于等于、等于和不等于。其中要注意等于运算符(==)和赋值运算符(=)的区别,初学者常犯的错误就是以赋值运算符(=)来表示相等的关系。

【例 3-2-9】 区别运算赋值=与关系运算相等==。

```
>>> 20 == 20
```

```
True
>>> 20 = 20
SyntaxError: can't assign to literal
>>> x,y = 10,20
>>> x == y
False
>>> x = y
>>> x
20
```

与其他高级语言不同,在 Python 中还允许使用级联比较形式,可用以下形式比较 a、b、c 三数的大小:$a \leqslant b \leqslant c$。

【例 3-2-10】 级联比较形式示例。

```
>>> a,b,c = 10,20,30
>>> a <= b <= c
True
```

对浮点数据进行相等的关系运算时,不能直接用等于(==)操作。浮点数类型能够表示巨大的数值,能够进行高精度的计算,但是由于浮点数在计算机内是用固定长度的二进制表示的,有些数可能没有办法精确地表示,计算会引起误差。

【例 3-2-11】 浮点数的误差示例。

```
>>> x = 3.141592627
>>> x - 3.14
0.0015926269999999576
```

上例 $x-3.14$ 的值并没有得到 0.001 592 627,结果略小一些,又如

```
>>> 2.1 - 2.0
0.10000000000000009
```

这个例子中得到的结果又比正确的结果略大了一些。从这个例子可以得到一条经验:不能用==来判断是否相等,而是要检查两个浮点数的差值是否足够小,是则认为是相等的。

```
>>> 2.1 - 2.0 == 0.1
False
>>> esp = 0.000000001
>>> abs((2.1 - 2.0) - 0.1)< esp
True
```

3) 逻辑运算

关系运算只能表示简单的布尔判断,复杂的布尔表达式还需要逻辑表达式来构成。逻辑表达式通过逻辑运算与(and)、或(or)、非(not),可以将简单的布尔表达式联结起来,构成更为复杂的逻辑判断,如表 3-2-2 所示。

- 逻辑与 and。

当计算 a and b 时,Python 会计算 a,如果 a 为假,则取 a 值,如果 a 为真,则 Python 会计算 b 且整个表达式会取 b 值。

表 3-2-2　逻辑运算的真值表

a	b	a and b	a or b	not a
False	True	False	True	True
False	False	False	False	True
True	True	True	True	False
True	False	False	True	False

- 逻辑或 or。

当计算 a or b 时，Python 会计算 a，如果 a 为真，则整个表达式取 a 值，如果 a 为假，表达式将取 b 值。

- 逻辑非 not。

如果表达式为真，not 返回假，如表达式为假，not 返回真。

【例 3-2-12】　判断某一年是否是闰年。

判断是否为闰年依据是否满足下面两个条件之一：

- 该年能被 4 整除但不能被 100 整数。
- 该年能被 400 整除。

```
>>> y = 2010
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
False
>>> y = 2012          # 符合第一个条件
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
True
>>> y = 2000          # 符合第二个条件
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
True
```

3. 表达式的求值

表达式的计算过程又称为表达式的求值。表达式可以很简单，也可以很复杂，其中包含了多个不同类型的运算符，那不同类型的运算符按照什么顺序运算呢？在数学表达式中的数据是不分类型的，都是数值，而计算机表达式中的数据区分不同的类型，同类型数据运算得到同类型的数据，那不同类型的数据出现在同一表达式中，如何运算？得到的是何种数据类型呢？毕竟不同数据类型的数据的存储编码形式是不同的。这就涉及表达式的计算顺序和混合运算的类型转换问题。

1）计算顺序

影响表达式计算顺序的因素包括：运算符的优先级、运算符的结合方式和括号。

（1）优先级。

小学生学算术的时候就知道先乘除后加减，也就是说乘除的优先级比加减要高，在计算中先做。程序设计语言会给每一个运算符确定一个优先级，具有较高优先级的运算符要比较低运算符优先计算。算术运算符的优先级设定与数学中基本相同。例如：下面表达式中的加法会最后做。

```
>>> 12/4 + 5 * 3/2
10.5
```

数值数据常用运算符的优先级由高到低如表 3-2-3 所示。

表 3-2-3 数值数据常用运算符的优先级

序号	运 算 符	描 述
1	$+x$, $-x$	正,负
2	$x**y$	幂
3	$x*y$, x/y, $x\%y$	乘,除,取模
4	$x+y$, $x-y$	加,减
5	$x<y$, $x<=y$, $x==y$, $x!=y$, $x>=y$, $x>y$	比较
6	not x	逻辑否
7	x and y	逻辑与
8	x or y	逻辑或

(2) 结合方式。

仅靠优先级,上面例子中 5 * 3/2 的计算方式还没有确定,因为相邻的乘运算和除运算的优先级是一样高的,结合方式或称结合律会规定具有相同优先级的运算符相邻出现时,运算符是从左向右结合,还是从右向左结合。例如,一目运算符是从右向左结合的,-2+3 中的负号是一目运算符,从右向左,操作数 2 为负数,二目运算符是从左向右结合的,上式中的加号,从左向右结合。5 * 3/2 的计算方式可确定先乘后除。这一规定也符合数学中的习惯。

(3) 括号。

括号可以突破计算的优先级,强制地规定计算顺序,括号括起部分的表达式会先行计算,计算的结果再参与括号外的表达式的计算。例如改写上例:12/(4+5) * 3/2,就可强制性先计算加法。

此外,运算对象还存在求值顺序问题。在一个较长的表达式中,不相邻的同级运算先算左边的对象还是右边的对象,例如:(34+22) * (3+5)式子中,两个括号应先行计算,但先计算(34+22),还是先计算(3+5),不同的程序语言,甚至不同的系统中有不同的规定。所以对运算对象的求值顺序可以这样理解:表达式的求值应不依赖于运算对象的求值顺序,因为它在各种系统中可能得到不同的顺序。运算对象求值顺序问题是程序语言中的特殊问题,在数学中不存在这种问题,这也是计算机与数学的不同。

2) 类型转换

计算机中的运算与数据类型有密切关系,由于数据类型的限制,程序中一般同类型数据运算得到同类型的数据,例如 3+4 得到 7,操作数和结果都是整数,但这一规则在下面式子的理解中会带来迷惑:

```
6/4 * 4
```

作为数学式子,结果很明显是 6。但在不同的程序设计语言中结果就不同了。在 C 语言中结果为 4,在 Python 语言中结果为 6.0。如何解释呢?

C 语言严格遵循两个 int 类型的数据计算,得到的还是 int 类型的原则,6/4 等于 1,1 乘以 4 等于 4。Python 语言将参加除法运算的操作数自动转化为 float 类型,再进行 float 类型的除法运算,6.0/4.0 等于 1.5,1.5 乘以 4.0 等于 6.0。

```
>>> 6/4 * 4
6.0
```

如果表达式中进行的是整除运算,得到的结果就与 C 语言中的相同了。

```
>>> 6//4 * 4
4
```

如果表达式中包含不同数据类型的数据对象,就出现了混合运算,任何运算都是定义于相同数据类型的,不同类型的数据运算要进行类型转换,只有同类型的数据对象才能进行运算。

表达式计算中碰到不同的数据类型例如 3.0+2,先通过转换将 2 转换为 2.0,然后才会进行实际的浮点数运算,这种转换是系统自动完成的,不需要在程序中写出,所以称为**自动转换**或**隐式转换**。

【例 3-2-13】 自动转换示例。

```
>>> 3.0 + 2
5.0
>>> type(3.0 + 2)
<class 'float'>
```

自动转换的基本原则是将表示数值范围小的数据类型的值转换到表示数值范围大的数据类型的值,这样能避免由于类型转换造成的误差损失。

如果自动转化不符合需求,程序语言还引入了**强制转换机制**或**显式转换**,在程序语句中明确类型转换的描述,要求执行类型转换。强制转换的出现形式是强制转换的运算,例如要将实数 3.14 转化为整数,Python 语言提供各种类型的转换函数,上式写作:int(3.14)。常用类型转换函数如表 3-2-4 所示。

表 3-2-4 常用类型转换函数

函　数	描　述	函　数	描　述
int(*x*[,base])	将 *x* 转换为一个整数	ord(*x*)	将一个字符转换为它的 ASCII 编码的整数值
float(*x*)	将 *x* 转换到一个浮点数	chr(*x*)	将一个整数转换为一个字符,整数为字符的 ASCII 编码
complex(real [,imag])	创建一个复数	hex(*x*)	将一个整数转换为一个十六进制字符串
str(*x*)	将对象 *x* 转换为字符串	oct(*x*)	将一个整数转换为一个八进制字符串
repr(*x*)	将对象 *x* 转换为字符串	eval(str)	将字符串 str 当成有效表达式来求值,并返回计算结果

【例 3-2-14】 显式转换示例。

```
>>> x,y = 23,12                #变量 x,y 的值为整数 23 和 12
>>> y = float(y) + 0.5         #强制转换 y 的值为 12.0 参加浮点运算
```

```
>>> y
12.5
>>> complex(x,y)              #创建复数,x,y 为实部和虚部的值
(23 + 12.5j)
>>> str(x)                    #读取 x 的值转化为字符串,存储在 x 中的值不变
'23'
>>> hex(x)                    #读取 x 的值转化为十六进制字符串,存储在 x 中的值不变
'0x17'
>>> oct(x)                    #读取 x 的值转化为八进制字符串,存储在 x 中的值不变
'0o27'
>>> repr(x + 20)              #读取 x 的值加 20 后转化为字符串,存储在 x 中的值不变
'43'
>>> chr(13)                   #得到 13 所表示的字符: Enter 键
'\r'
>>> ord('\n')                 #得到换行符的 ASCII 值
10
>>> eval('x - y')             #计算字符串表示的表达式的值
11
```

3.2.3 系统函数

除了使用运算符进行运算,一般的高级语言程序系统中都提供系统函数丰富的语言功能。Python 提供模块的方式,可方便地扩充语言的功能。Python 的系统函数由标准库中的很多模块提供。标准库中的模块,又分成内置模块和非内置模块,内置模块__builtin__中的函数和变量可以直接使用,非内置模块要先导入模块,再使用。

1. 内置模块

Python 中的内置函数是通过__builtin__模块提供的,该模块不需要手动导入,启动 Python 时系统会自动导入,任何程序都可以直接使用它们。该模块定义了一些软件开发中常用的函数,这些函数实现了数据类型的转换、数据的计算、序列的处理、常用字符串处理等。

函数的调用方式与数学函数类似,函数名加上相应的参数值,多个参数值之间以逗号分隔。

<函数名>(参数序列)

【例 3-2-15】 内置模块函数示例。

```
>>>### help(obj) 在线帮助, obj 可以是任何类型,例如查看 math 模块的内容
>>>help(math)

>>> ##int("123")可将字符串"123"转换为整数 123
>>> int("123")
123
>>> ## int(78.9)得到整数 78(去掉尾部小数)
>>> int(78.9)
78
>>>## reper(obj),将任意值转为字符串,常用于构造输出字符串
>>> x = 10 * 3.25
```

```
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print( s )
The value of x is 32.5, and y is 40000...
>>> ##使用round(x,n)可按"四舍五入"法对x保留n位小数
>>> round(78.3456,2)
78.35
>>>##使用len(s)计算字符串的长度
>>> len("Good morning")
12
```

2. 非内置模块

1）非内置模块的导入

非内置模块在使用前要先导入模块，Python中使用以下语句来导入模块：

import <模块名>

其中模块名也可以有多个，多个模块之间用逗号分隔。该语句通常放在程序的开始部分。模块导入后，可以在程序中使用模块中定义的函数或常量值。

<模块名>.<函数>(<参数>)
<模块名>.<字面常量>

【例3-2-16】 以数学库为例。

```
>>> import math                 ##导入数学库
>>> math.pi                     ##查看圆周率π常数
3.141592653589793
>>> math.pow(math.pi,2)         ## 函数pow(x,y): 求x的y次方
9.869604401089358
>>>##计算边长为8.3和10.58,两边夹角为37°的三角形的面积的表达式为
>>> 8.3 * 10.58 * math.sin(37.0/180 * math.pi)/2
26.423892221536985
```

数学库中函数引入和使用的另外一种方式。

【例3-2-17】 数学库中函数引入和使用的另外一种方式。

```
>>> from math import sqrt #引入数学库中的sqrt函数
>>> sqrt(16)
4.0
```

如果希望导入math模块中所有的函数定义，而非仅仅是sqrt函数可以使用以下形式：

```
>>> from math import *      #引入数学库中所有的函数
>>> sqrt(16)
4.0
```

注意：引入方式不同，对应的函数的使用方式不同，还要注意所引入模块中的函数名等与现有系统中不产生冲突。

2）常用的标准数学函数

常用的标准数学函数包括三角函数、反三角函数、指数函数、对数函数，平方根函数、绝对值函数和乘幂函数，如表3-2-5所示。

表 3-2-5　math 库中的常用函数和字面常量

常用函数	描　　述	常用函数	描　　述
pi	常数 π(近似值)	sin(x)	正弦函数
e	常数 e(近似值)	cos(x)	余弦函数
fabs	求绝对值	tan(x)	正切函数
trunc(x)	将一个浮点数结尾为整数	ceil(x)	大于等于 x 的最小整数
factorial(x)	求 x 的阶乘	floor(x)	小于等于 x 的最大整数
pow(x,y)	求 x 的 y 次方		
sqrt(x)	求 x 的平方根		

3.3　文本数据的表示和操作

计算机早期是应科学计算的需求而产生的，程序的处理对象是数值型的数据。随着计算机的应用在各行各业的普及，程序的处理对象也日益丰富，大量应用于文本数据的处理，例如各类信息管理系统、文本编辑器、电子出版物、搜索引擎等。文本数据在程序中通常是以字符串类型表示的，字符串由字符构成。

Python 语言表示字符串的数据类型是 str 类，str 类定义了字符串类型的常量表示、基本运算和操作方法。

```
>>> type("shanghai")
<class 'str'>
```

3.3.1　文本的表示

1. 字符

计算机中表示文本的最基本的单位是字符，包括可打印字符和不可打印的控制字符。可打印字符包括：

(1) 英文的大小写字母 a～z，A～Z；

(2) 数字字符 0～9；

(3) 标点符号和一些键盘上的常见符号。

控制字符包括回车、制表符、退格等，在程序中需要以转义字符表示这些控制字符。Python 中的转义字符以\为前缀，如表 3-3-1 所示。

表 3-3-1　Python 的转义字符

转义字符	描　　述	转义字符	描　　述
\\	反斜杠符号	\t	横向制表符
\'	单引号	\r	回车
\"	双引号	\n	换行
\a	响铃	\(在行尾时)	续行符
\b	退格(Back Space)	\f	换页
\e	转义	\oyy	八进制数 yy 代表的字符 例如：\o12 代表换行
\000	空	\xyy	十六进制数 yy 代表的字符 例如：\x0a 代表换行

2. 字符串常量

字符串可以使用双引号或单引号封装,但前后必须一致。

【例 3-3-1】 字符串常量表示。

"hello"和'hello'表示的都是字符串 hello。

```
>>> "hello"
'hello'
>>> 'hello'
'hello'
```

如果字符串本身要带引号使用转义字符\"。

```
>>> '\"hello\"'
'"hello" '
>>> '\'hello\''
" 'hello'"
```

Python 还支持只有引号的空字符串。

```
>>> ""
''
```

Python 同样支持以\为前缀的转义字符,例如使用转义字符\n 可以在输出时使字符串换行。

```
>>> print("hello everyone\ntoday is a great day!")
hello everyone
today is a great day!
```

3. 字符串变量

字符串同样也可以使用字符串变量来操作。

【例 3-3-2】 字符串变量示例。

```
>>> s = "hello"
>>> print(s)
hello
```

字符串变量的实质是一个指向字符串对象的标识符。

【例 3-3-3】 字符串对象标识示例。

通过赋值语句生成两个字符串变量 a 和 b。

```
>>> a = "shanghai"
>>> b = "china"
```

使用内置函数 id(对象)可以得到对象 ID,例如 a 的对象 ID 为 35 546 720,b 的对象 ID 为 35 467 616。

```
>>> id(a)
35546720
>>> id(b)
35467616
```

执行赋值语句 $a=b$ 后,a 指向 b 所指向的字符串对象 35 467 616,a 原来指向的字符串

对象 35 546 720 的存储单元通过系统的垃圾回收机制由内存回收。

```
>>> a = b
>>> id(a)
35467616
```

3.3.2 字符串类型数据的基本计算

1. 联接和复制操作

字符串类型支持的运算有+和 *,可以使用+联接两个字符串。

【例 3-3-4】 联接运算示例。

```
>>> 'shang' + 'hai'
'shanghai'
```

【例 3-3-5】 复制运算示例。

运算 * 可以生成重复字符串,用法是[字符串] * [整数],非常方便,例如:

```
>>> "hi " * 5
'hi hi hi hi hi '
>>> s = "hi"
>>> t = s * 3
>>> print(t)
hihihi
```

2. 索引操作

使用方括号来获取字符串中指定的某个字符,用法是:

```
<字符串>[<数值表达式>]
```

【例 3-3-6】 字符串索引示例。

```
>>> "Student"[5]
'n'
>>> s = "hello Python!"
>>> s[0]
'h'
>>> i = 10
>>> s[i + 1]
'n'
>>> s[ - 1]
'!'
```

注意:Python 中索引位置是从 0 开始计数的,数值表达式可以为负数,则表示从右向左计数,字符串中的字符引用,*s* 的合法下标标识如图 3-3-1 所示。

h	e	l	l	o		p	y	t	h	o	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

图 3-3-1 字符串的下标示例

注意：Python 不支持以任何方式改变字符串类型对象的值，不能通过索引的方式来改变字符串中的某一个字符。

【例 3-3-7】 字符串修改错误示例。

```
>>> s[5] = 'i'
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    s[5] = 'i'
TypeError: 'str' object does not support item assignment
```

出错提示给出类型错误：str 对象不支持对其成员赋值。

Python 还支持通过索引操作获取字符串的子串，也称为切片操作，用法是指定子串的区间，start 表示开始位置，end 表示结束位置（下标−1）。

```
<字符串>[ start: end]
```

【例 3-3-8】 子串索引示例。

```
>>> s[0:2]
'he'
>>> s[2:4]
'll'
>>> s[:2]      # 前面的两个字符
'he'
>>> s[2:]      # 除了开始 2 个字符的所有字符
'llo Python'
```

3. 子串测试操作

子串测试操作 in 可以测试一个子串是否存在于一个字符串中，计算返回布尔值，用法为：

```
<子串> in <字符串>
```

【例 3-3-9】 子串测试操作示例。

```
>>> 'py' in s
True
>>> t = 'the'
>>> t in s
False
```

3.3.3 str 对象的方法

str 类提供了丰富的字符串操作方法，如表 3-3-2 所示，S 表示一个 str 对象。读者同样可以通过 help(str)查询更多的字符串操作的方法。

表 3-3-2 str 对象的常用方法

str 的常用方法	描 述
S. capitalize()	返回首字符大写后的字符串，S 对象不变
S. lower()	返回所有字符改小写后的字符串，S 对象不变
S. upper()	返回所有字符改大写后的字符串，S 对象不变
S. strip()	返回删去前后空格后的字符串，S 对象不变

续表

str 的常用方法	描　　述
S. replace(old,new)	将 S 对象中所有的 old 子串用 new 子串代替
S. count(sub[,start[,end]])	计算子串 sub 在 S 对象中出现的次数,start 和 end 定义起始位置
S. find(sub[,start[,end]])	计算子串 sub 在 S 对象中首次出现的位置
S. join(iterable)	将序列对象中所有字符串合并成一个字符串,S 对象为连接分隔符
S. split(sep=None)	将 S 对象按分隔符 sep 拆分为字符串列表,默认为空格

str 对象方法的调用形式为

<字符串>. 方法名(<参数>)

如图 3-3-2 所示,在命令行提示符后输入对象名,稍作停留,会显示该对象所有方法的列表,使用上下光标键可以选择所需的方法。

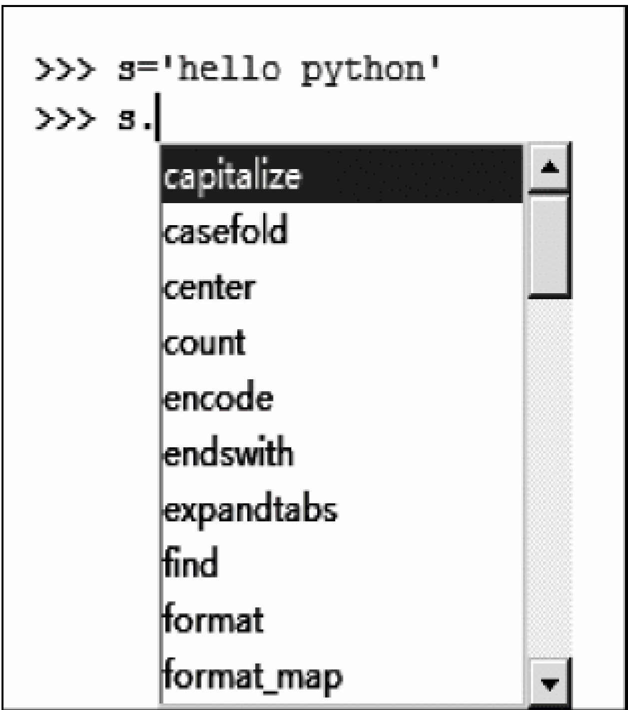


图 3-3-2　弹出的对象的方法列表示例

【例 3-3-10】 str 对象方法示例。

```
>>> s = "hello Python!"
>>> s.find('he')   #求子串'he'第一次出现的位置
0
>>> s.count('h')   #求'h'出现的次数
2
```

使用 str 类的方法同样不能改变字符串对象的值,例如调用 strip 函数去除字符串的前后空格,它的作用是返回一个去除了原字符串的前后空格的新串,原字符串不变。

```
>>> s = '   hello Python   '
>>> t = s.strip()
>>> s
'   hello Python   '
>>> t
'hello Python'
```

同样 lower、upper、replace 等会产生字符修改的函数,都是返回一个新的字符串对象,而字符串对象本身的内容是不变的。

【例 3-3-11】 字符串对象的连接和分裂操作示例。

```
>>> a = ["hello","Python"]       # a 为一个包含两个字符串的列表
>>> b = " ".join(a)              #将 a 中的两个字符串连接,并以空格分隔,赋值给 b
>>> b
'hello Python'
>>> b.split()
['hello', 'Python']
>>> c = ",".join(a)
>>> c
'hello,Python'
>>> c.split(',')
['hello', 'Python']
```

3.4 批量数据表示与操作

3.4.1 批量数据的构造

1. 批量数据的概念

在实际的计算机处理问题中，程序要处理的对象都是大批量的相同数据类型的数据集合，例如一次科学实验中获得的大量实验数据，关键字搜索时大量的网页中所包含的单词，一幅 BMP 图像中包含的像素点。这几个例子中分别包含大量数值的集合、大量文本的集合和大量点对象的集合。程序语言支持批量数据的存储，用统一的名称管理一批数据，在内存的存储上表现为存储的空间是连续的。

对批量数据中元素的访问可通过下标。例如：$a[1]$，$a[i]$。下标的含义：与第一个元素的偏移量，通常从 0 开始。

例如有：

```
Color = ("red","green","blue")
```

则 Color[0]的值是"red"，Color[1]的值是"green"，Color[2]的值是"blue"，内存示意如图 3-4-1 所示。

Color[0]	"red"
Color[1]	"green"
Color[2]	"blue"

图 3-4-1　Color 值的内存示意图

批量数据的存储与单变量数据存储相比的优势在于：

(1) 一批数据只需定义一个名称，程序的通用性更强。一个单变量只可以控制一个数据，使用单变量，程序可控制的数据的个数是固定的。

(2) 使用方便，可以组织循环控制结构，通过控制下标的值控制一批数据。

大多数程序设计语言提供了数组来组织批量数据的存储与操作，一个数组中存储着具有相同数据类型的数据，通过下标引用其中的每一个数组元素，数组元素可以实现该数据类型所定义的运算。面向对象的程序语言还提供了功能更为强大的列表类、向量类等，在定义批量对象的存储之外，同时提供对批量数据的常规操作。

2. Python 对批量数据的表示和操作

Python 可支持批量数据存储和操作的组合数据类型有列表(list)、元组(tuple)、字典(dict)、集合(set)等。字符串也可以看作字符的组合类型。

这些数据类型按数据是否是在内存中连续排列的集合体可区分为两种方式：有序的数据集合体和无序的数据集合体。

有序的数据集合体，也称为序列，包括字符串、元组和列表。序列的数据成员之间存在排列次序，因此可以通过各数据成员在序列中所处的位置，即索引或下标来访问数据成员。Python 提供序列的一些通用操作，如表 3-4-1 所示，以实现对序列的索引、联接、复制、检测成员等。

表 3-4-1　序列的基本操作

操　　作	描　　述
$x1+x2$	联接序列 $x1$ 和 $x2$，生成新序列
$x*n$	将序列 x 复制 n 次，生成新序列
$x[i]$	引用序列 x 中下标为 i 的序列成员，i 从 0 开始计数

续表

操　　作	描　　述
$x[i:j]$	引用序列 x 中下标从 i 到 $j-1$ 的子序列
$x[i:j:k]$	引用序列 x 中下标从 i 到 $j-1$，间隔为 k 的子序列
len(x)	计算序列 x 中成员的个数
max(x)	序列 x 中最大数据项
min(x)	序列 x 中最大数据项
v in x	检测 v 是否存在序列 x 中，返回布尔值
v not in x	检测 v 是否不存在序列 x 中，返回布尔值

无序的数据集合体包括集合、字典等，无序的数据集合体中的数据成员之间不存在存储的先后关系，故也不支持索引操作。

3. 类型构造器

在 Python 语言中，所有事物都是程序可以访问的对象，所有表示数据的类型都是类(class)，包括简单数据类型如 int、float、bool、complex 和 Python 的组合数据类型如列表(list)、元组(tuple)、字典(dict)、集合(set)。这一点可以从 type 函数示例中看到。

每一个类都提供了类型构造器，类型构造器是一个与类同名的函数。例如表 3-2-4 中列出的 int()、float()、str()都是类型构造器，它们通常可以生成一个空的对象，将一个对象转换为本类对象。同样本节涉及的容器类型对象都有各自的类型构造器函数，完成创建对象、转换对象的功能。类型构造器的工作原理将在第 8 章介绍。

【例 3-4-1】 类型构造器示例。

生成空字符串对象。

```
>>> s1 = str()
>>> s1
''
```

将一个学号(整数类型)转化为字符串。

```
>>> s2 = str(101030311245)
>>> s2
'101030311245'
```

3.4.2 元组和列表

Python 中的元组和列表是可以存储任意数量的一组相关数据形成一个整体，其中的每一项可以是任意类型的数据项。各数据项之间按索引号排列并允许访问。

元组和列表的区别为：元组是固定的，创建之后就不能改变其数据项；而列表创建后允许修改、插入或删除其中的数据项。

1. 元组的基本操作

1）元组的字面表示

元组一般使用圆括号来表示，数组项之间用逗号分隔。

【例 3-4-2】 字面表示方式创建元组。

```
>>> t = 1,2,3
>>> t
(1, 2, 3)
>>> t1 = (1,2,3)
>>> t1
(1, 2, 3)
```

数据项可以是相同类型的,也可以是不同类型的。

```
>>> t2 = "east","south","west","north"
>>> t2
('east', 'south', 'west', 'north')
>>> t3 = "0010110","张山","men",18
>>> t3
('0010110', '张山', 'men', 18)
```

可以定义空的元组,也可以定义嵌套的元组。

```
>>> t4 = ()
>>> t4
()
>>> t5 = 23,(5,8,6),18,6
>>> t5
(23, (5, 8, 6), 18, 6)
>>> t6 = t1,t2,t3,t4,t5
>>> t6
((1, 2, 3), ('east', 'south', 'west', 'north'), ('0010110', '张山', 'men', 18), (), (23, (5, 8, 6),
18, 6))
```

2) 元组的类型构造器

元组的类型构造器 tuple()可以生成一个空的元组,也可以将字符串、列表、集合等转化为元组。可以想象,容器类型对象可以通过它们的类型构造器相互转化。

【例 3-4-3】 元组的类型构造器示例。

生成一个空的元组。

```
>>> t7 = tuple()
>>> t7
()
```

将一个字符串转化为元组。

```
>>> t7 = tuple('Python')
>>> t7
('P', 'y', 't', 'h', 'o', 'n')
```

3) 元组元素的访问

可以通过下标访问元组中的某一项,成为元组元素。同样下标从 0 开始,但不能修改元组中的元素。

【例 3-4-4】 元组元素的访问示例。

```
>>> t6[2]
('0010110', '张山', 'men', 18)
>>> t6[2][0]
'0010110'
>>> t6[2] = t2
Trace  back (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    t6[2] = t2
TypeError: 'tuple' object does not support item assignment
```

2. 列表的基本操作

1）列表的字面表示

列表的创建与元组的区别在于需要使用方括号。其他与元组类似,数据项之间以逗号分隔,可以嵌套定义,可以是不同的数据类型,可以是空列表。可以用下标访问其中的数据项。

【例 3-4-5】 字面表示方式创建元组。

```
>>> L1 = ["one","two","three","four","five"]                    #由5个字符串构成的列表
>>> L2 = [[1,2],[3,4],[5,6]]                                    #由3个列表构成的嵌套列表
>>> L3 = ["zhangsan",True,185,"lisi",False,165,"wanger",True,176]  #混合类型的列表
>>> L4 = [[],[],[]]                                                #嵌套空列表的列表
>>> L5 = []                                                        #生成空的列表
```

2）列表的类型构造器

列表的类型构造器 list()可以生成一个空的列表,也可以将字符串、元组、集合等转化为元组。

【例 3-4-6】 列表的类型构造器示例。

```
>>> L5 = list()                   #生成空的列表,与L5 = []功能相同
>>> L6 = list('Pyhton')           #将一个字符串对象转化为列表
>>> L6
['P', 'y', 'h', 't', 'o', 'n']
>>> L7 = list(('he','her','here')) #将一个元组转化为列表
>>> L7
['he', 'her', 'here']
```

3）列表元素的访问

列表同样支持索引访问,访问特定列表元素,或是子列表,修改列表元素。

【例 3-4-7】 列表元素的访问。

```
>>> L1[1]
'two'
>>> L2[2][1]
6
```

列表和元组的根本区别在于可以改变列表中的元素。

【例 3-4-8】 修改指定位置的元素。

```
>>> L2[2] = 5
>>> L2
[[1, 2], [3, 4], 5]
>>> L2[0][0] = L2[0][1] * 10
>>> L2
[[20, 2], [3, 4], 5]
```

【例 3-4-9】 联接列表元素。

```
>>> L2 = L2 + [7,8]                    #将两个列表的表项联接为一个列表
>>> L2
[[20, 2], [3, 4], 5, 7, 8]
>>> L2 = L2 + [[9,10]]
>>> L2
[[20, 2], [3, 4], 5, 7, 8, [9, 10]]
```

【例 3-4-10】 在指定位置插入数据项。

```
>>> L2[3:3] = [6]                      #3:3 表示下标为 3 的位置,在此位置插入列表[6]的表项 6
>>> L2
[[20, 2], [3, 4], 5, 6, 7, 8, [9, 10]]
```

【例 3-4-11】 删除指定位置的数据项。

```
>>> L2[2:6] = []          #引用 L2 中 2～5 的表项,将子序列通过赋值操作更改为空序列
>>> L2
[[20, 2], [3, 4], [9, 10]]
```

【例 3-4-12】 在指定位置插入嵌套数据项。

```
>>> L2[2:2] = [[5,6],[7,8]]
>>> L2
[[20, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
```

【例 3-4-13】 可以使用 Python 内置函数 len(),来计算元组或列表的长度。

```
>>> len(L2)
5
```

【例 3-4-14】 使用 in 和 not in 来测试是否是元组或列表成员,测试结果返回布尔值(True 或 False)。

```
>>> 'wanger' in L3
True
>>> t2 in t6
True
>>> 7 in L2
False
>>> [7,8]in L2
True
```

3. 元组对象方法和列表对象的方法

由于元组对象创建后不能改变自身的值,是只读属性的对象,它的方法只有两个,如

表 3-4-2 所示,T 表示一个元组对象。

表 3-4-2 对象的常用方法

方　　法	描　　述
T. count(value)	计算 value 值在元组中出现的次数
T. index(value)	计算 value 值在元组中出现的下标

相对于元组对象的方法,列表的方法就丰富得多,如表 3-4-3 所示,L 表示一个列表对象。

表 3-4-3 列表对象的常用方法

方　　法	描　　述
L. append(object)	在列表 L 尾部追加对象
L. clear()	移除列表 L 中的所有对象
L. count(value)	计算 value 在列表 L 中出现的次数
L. copy()	返回 L 备份的新对象
L. extend(Lb)	将 Lb 的表项扩充到 L 中
L. index(value, [start, [stop]])	计算 value 在列表 L 指定区间第一次出现的下标值
L. insert(index, object)	在列表 L 的下标为 index 的表项前插入对象
L. pop([index])	返回并移除下标为 index 的表项,默认最后一个
L. remove(value)	移除第一个值为 value 的表项
L. reverse()	倒置列表 L
L. sort()	对列表中的数值按从低到高的顺序排序

使用列表对象的方法,可以很方便地存储、维护、分析批量数据。

【例 3-4-15】 对某居民家庭一年的用电情况进行维护和分析。

(1) 初始化列表。

```
>>> t = [ ]
```

(2) 增加 1 月份的用电量。

```
>>> t.append(271)
```

(3) 批量增加 2～12 月份的用电量。

```
>>> t.extend ([151,78,92,83,134,357,421,210,88,92,135])
>>> t
[271, 151, 78, 92, 83, 134, 357, 421, 210, 88, 92, 135]
```

注意:append 方法是将一个对象追加到列表中,append 方法的参数是一个任意对象,作为一个表项加入列表中;extend 方法是将一个列表中的表项扩充到列表中去,所以 extend 方法的参数是一个列表,参数列表的表项加入列表中。

(4) 修改 8 月份的用电量 421 为 425。

```
>>> t.remove(421)
>>> t.insert(7,425)
>>> t
[271, 151, 78, 92, 83, 134, 357, 425, 210, 88, 92, 135]
```

注意：修改列表表项的方法还可以直接通过索引访问：$t[7]=425$，更为方便。在此只演示 move 和 insert 方法的使用。

(5) 求用电量最高的月份 maxm。

解题思路：先计算 t 中的最大值，再寻找最大值在 t 中出现的位置，下标从 0 开始，加 1 就是对应的月份。求最大值使用 max 函数实现，寻找一个数值在列表中出现的位置，使用 index 方法实现。

```
>>> maxm = t.index(max(t)) + 1
>>> maxm
8
```

(6) 按用电量从低到高排序。

```
>>> s = t.copy()
>>> s.sort()
>>> s
[78, 83, 88, 92, 92, 134, 135, 151, 210, 271, 357, 425]
```

注意：这里不能直接用 $s=t$，而是要使用 copy 函数得到一个副本对象。$s=t$ 的含义是 s 指向 t 对象，那么对 s 排序就等同于对 t 排序了。如果想得到从高到低的排序结果，可以增加一个参数值设定：s.sort(reverse=True)。

(7) 找出用电量最高的三个月。

解题思路：s 序列是 t 序列从低到高的有序序列，倒序后，序列值从高到低排列，序列的前三项就是用电量最高的三个值。再寻找三个值在源序列 t 中出现的位置，加 1 后就可以计算出月份。

```
>>> s.reverse()
>>> m1,m2,m3 = t.index(s[0]) + 1,t.index(s[1]) + 1,t.index(s[2]) + 1
>>> print(m1,m2,m3)
8 7 1
```

结合第 4 章所讲的控制结构，可以对居民家庭的用电情况做更为复杂的分析统计。

3.4.3 集合和字典

1. 集合

Python 的集合也是一个内置的数据类型，与列表和元组不同的是集合是无序的，而且集合的元素不能重复出现，不能通过数字进行索引。正是因为它具有的这些特性，所以通常可用来进行一些数据中转处理，如去除列表中的重复元素(集体元素是唯一的)，两个列表的相同元素(交集)等。

1) 创建集合

Python 的集合可分为可变集合(set)和不可变集合(frozenset)。对可变集合(set)，可以添加和删除元素，对不可变集合(frozenset)则不允许这样做。可变集合可以通过集合标识符{}直接创建，也可以通过类型构造器 set()创建，不可变集合需要通过类型构造器 frozenset()创建。

使用{}创建的是可变集合 set，{}中用逗号分隔的数据项作为集合的一个元素。

【例 3-4-16】 集合的字面表示示例。

```
>>> s1 = {2,4,6,8,10}
>>> type(s1)
<class 'set'>
>>> s1
{8, 10, 4, 2, 6}
>>> s2 = {'hello'}
>>> s2
{'hello'}
```

set 函数的参数是容器对象,可以是字符串,列表和元组,它可以将序列的数据元素作为集合 set 的元素。

【例 3-4-17】 集合的类型构造器示例。

```
>>> s3 = set('hello')
>>> s3
{'l', 'e', 'o', 'h'}
```

字符串 hello,由 5 个字符构成,其中 l 出现了两次,转换到集合中,重复项只能保留一个,且字符次序与原字符串的次序不同。集合的这种特性,可以很方便地对列表对象执行去重复的功能,同样还可以根据列表对象来创建集合。

```
>>> s5 = set(['he','hello','her','here'])
>>> s5
{'here', 'hello', 'he', 'her'}
```

【例 3-4-18】 列表去重复操作示例。

通过 set 函数建立列表的去重复集合元素,再通过 list 方法根据集合创建列表。

```
>>> L1  = [1,2,3,4,1,2,3,4]
>>> s4 = set(L1)
>>> s4
{1, 2, 3, 4}
>>> L2 = list(s4)
>>> L2
[1, 2, 3, 4]
```

L2 是去重复后的列表,上面的过程也可简单地写为

```
>>> L2 = list(set(L1))
>>> print L2
[1, 2, 3, 4]
```

set 是可以改变的集合类型,如果创建后的集合元素不需要改变,可创建不可变集合。

【例 3-4-19】 创建一个星期的英文缩写的不可变集合。

```
>>> s6 = frozenset(('MON','TUE','WED','THU','FRI','SAT','SUN'))
>>> s6
frozenset({'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'})
```

2）访问集合

由于集合本身是无序的，所以不能为集合创建索引或切片操作，只能循环遍历或使用in、not in 来访问或判断集合元素。

【例 3-4-20】 集合访问示例。

```
>>> 'SUN' in s6
True
>>> 'SON' in s6
False
>>> for i in s6:
        print(i,end = " ")

SUN WED TUE SAT FRI MON THU
```

3）集合运算

集合支持的运算有：交集、并集、差集、对称差集，和中学数学中学习的集合运算的概念相同，常用的集合运算如表 3-4-4 所示。

表 3-4-4　集合的常见运算

运　算	描　述	运　算	描　述
x in $s1$	检测 x 是否在集合 $s1$ 中	$s1==s2$	判断集合是否相等
$s1\|s2$	并集	$s1<=s2$	判断 $s1$ 是否是 $s2$ 的子集
$s1\&s2$	交集	$s1<s2$	判断 $s1$ 是否是 $s2$ 的真子集
$s1-s2$	差集	$s1>=s2$	判断 $s1$ 是否是 $s2$ 的超集
$s1$^$s2$	异或集，求 $s1$ 与 $s2$ 中相异元素	$s1>s2$	判断 $s1$ 是否是 $s2$ 的真超集
$s1\|=s2$	将 $s2$ 的元素并入 $s1$		

【例 3-4-21】 集合运算示例。

对前面已建立的集合 $s2$、$s5$ 做以下操作：

```
>>> s2
{'hello'}
>>> s5
{'here', 'hello', 'he', 'her'}
>>> s2 <= s5            # 判断 s2 是否是 s5 的子集
True
>>> s5 > s2             # 判断 s5 是否是 s2 的真超集
True
```

创建新的集合 $s7$，做以下操作。

```
>>> s7 = {'hen','height','her'}
>>> s7 |= s2            # 将 s2 并入 s7
>>> s7
{'hello', 'her', 'height', 'hen'}
>>> s7&s5               # 求 s7 和 s5 的交集
{'hello', 'her'}
>>> s7|s5               # 求 s7 和 s5 的并集
{'here', 'her', 'hello', 'hen', 'he', 'height'}
```

```
>>> s7 - s5              # 求 s7 中去除 s5 中有的元素
{'height', 'hen'}
>>> s7 ^ s5              # 求 s7 和 s5 的中各不相同的元素
{'he', 'hen', 'height', 'here'}
```

判断两个集合是否相等,只需判断其中包含的集合元素是否一致,与顺序无关,下面的例子又说明了集合是无序的。

```
>>> s8 = {'he','hello', 'her','here'}
>>> s8
{'her', 'hello', 'he', 'here'}
>>> s5
{'here', 'hello', 'he', 'her'}
>>> s5 == s7
True
```

4) 集合对象的方法

Python 以对象方式实现集合类型,假设 *s*1 是集合对象,*s*2 可以是一个可迭代对象,集合对象的常用方法如表 3-4-5 所示,可以支持可变集合完成集合元素的增加、删除和集合的复制。

表 3-4-5　集合对象的常用方法

	方　　法	描　　述
	*s*1. union(*s*2)	*s*1|*s*2,返回一个新的集合对象
	*s*1. difference(*s*2)	*s*1−*s*2,返回一个新的集合对象
	*s*1. intersection(*s*2)	*s*1&*s*2,返回一个新的集合对象
	*s*1. issubset(*s*2)	*s*1<=*s*2
	*s*1. issuperset(*s*2)	*s*1>=*s*2
*	*s*1. update(*s*2)	将 *s*2 的元素并入 *s*1
*	*s*1. add (*x*)	增加元素 *x* 到 *s*1
*	*s*1. remove(*x*)	从 *s*1 移除 *x*,*x* 不存在报错
*	*s*1. clear ()	清空 *s*1
	*s*1. copy()	复制 *s*1,返回一个新的集合对象

其中打星号 * 的方法是 set 集合独有的方法,不打星号的方法是 set 和 frozenset 两种集合都有的方法。

【例 3-4-22】 集合对象的方法示例。

set 和 frozenset 对象都有 union 方法,返回一个合并后的新对象(调用该方法的原对象内容不变),要注意的是 set 对象调用 union 方法,返回的是 set 对象,frozenset 对象调用 union 方法,返回的是 frozenset 对象。

```
>>> s5                   # set 对象
{'here', 'hello', 'he', 'her'}
>>> s6                   #  frozenset 对象
frozenset({'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'})
>>> s9 = s6.union(s5)  #  frozenset 对象调用 union 得到 frozenset 对象
>>> s9
```

```
frozenset({'her', 'SUN', 'hello', 'WED', 'here', 'TUE', 'SAT', 'FRI', 'MON', 'he', 'THU'})
>>> s10 = s5.union(s6)          # set 对象调用 union 得到 set 对象
>>> s10
{'her', 'SUN', 'MON', 'WED', 'here', 'TUE', 'SAT', 'FRI', 'hello', 'he', 'THU'}
>>> s5                          #调用 union 产生新对象,原对象不变
{'here', 'hello', 'he', 'her'}
>>> s10 = s10.difference(s5)    #返回 s10 和 s5 的差集,产生一个新对象,再赋值给 s10
>>> s10
{'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'}
```

注意：union 方法是返回一个新的对象，调用对象本身不发生变化。update 方法是对调用方法的对象直接修改，所以只适合可修改的 set 对象，表 3-4-5 中打星号 * 的方法都会修改调用方法的对象本身。

```
>>> s5.update(s6)       #将 s6 的元素并入 s5 中
>>> s5
{'MON', 'FRI', 'he', 'THU', 'here', 'WED', 'her', 'hello', 'SUN', 'SAT', 'TUE'}
```

表中的 s2 并不要求是相同类型的对象，只要是一个可迭代(iterable)的对象，包括字符串、列表、元组、集合，例如：

```
>>>##将一个列表与集合 s6 联合
>>> L1 = [1,2,3]
>>> s11 = s6.union(L1)
>>> s11
frozenset({1, 2, 3, 'TUE', 'SAT', 'FRI', 'SUN', 'MON', 'WED', 'THU'})
>>>##判断 1 是否在 s11 集合中不能用 int 类型,要用列表类型或集合类型
>>> s11.issuperset(1)
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    s11.issuperset(1)
TypeError: 'int' object is not iterable
>>> s11.issuperset({1})
True
>>> s11.issuperset([1])
True
```

5）应用

可以利用集合运算很方便地比对两个集合的相同元素和不同元素。

【例 3-4-23】 利用集合分析活动投票情况。

两个小队举行活动评测投票，按队员序号投票，第一小队队员序号为 1、2、3、4、5，第二小队队员的序号为 6、7、8、9、10，可以对投票数据进行分析，投票数据为 1，5，9，3，9，1，1，7，5，7，7，3，3，1，5，7，4，4，5，4，9，5，5，9

建立集合 *s*2 表示第一小队队员序号，*s*3 表示第二小队队员序号。

```
>>> s2 = {1,2,3,4,5}
>>> s3 = {6,7,8,9,10}
```

使用投票数据建立集合 *s*1，集合去重复后表示获得了选票的队员序号。

```
>>> s1 = {1,5,9,3,9,1,1,7,5,7,7,3,3,3,1,5,7,4,4,5,4,9,5,5,9}
>>> s1
{1, 3, 4, 5, 7, 9}
```

第一小队获得选票的队员有

```
>>> s1 - s3
{1, 3, 4, 5}
```

第一小队没有获得选票的队员有

```
>>> s2 - (s1 - s3)
{2}
```

第二小队获得选票的队员有

```
>>> s1 - s2
{9, 7}
```

第二小队没有获得选票的队员有

```
>>> s3 - (s1 - s2)
{8, 10, 6}
```

2. 字典

序列采用查找信息的方式是通过序列元素的位置下标引用指定的序列元素，字典采用了另一种通过键值来查找信息的方式，键值和索引值反应了一种数据之间的关联，例如在表示星期时，通常用 1 表示星期一(MON)，6 表示星期六(SAT)，0 表示星期日(SUN)。字典是一个由键和值组成的键值对构成的集合，每一个字典元素分为两部分：键(key)和值(value)。

字典是 Python 中唯一内置映射数据类型，可以通过指定的键从字典访问值。字典类型 dict 与集合类型 set 一样是无序的集合体，键值对没有特定的排列顺序，所以不能通过位置下标访问字典元素。

1) 字典的创建

字典的创建同样可以通过字面值和类型构造器的方式。

• 字面值。

字典的字面值是由一对花括号括起的，以逗号分隔的键值对构成，键值对的书写形式为<键>：<值>

【例 3-4-24】 字典的字面表示示例。

```
>>> d1 = {1:'MON',2:'TUE',3:'WED',4:'THU',5:'FRI',6:'SAT',0:'SUN'}
>>> d1
{0: 'SUN', 1: 'MON', 2: 'TUE', 3: 'WED', 4: 'THU', 5: 'FRI', 6: 'SAT'}
```

与集合类似，字典中键值对的顺序与定义时的顺序是不一样的。

字典可嵌套，可以在一个字典里包含另一个字典。

【例 3-4-25】 嵌套字典示例。

```
>>> test = {"test":{"mytest":10} }
>>> test
{'test': {'mytest': 10}}
```

• 类型构造器 dict()。

使用类型构造器构造字典,参数为键值对,键值对之间以“,”分割,键值对的书写形式为<键>=<值>。

【例 3-4-26】 字典的类型构造器构造示例。

```
>>> monthdays = dict( Jan = 31, Feb = 28, Mar = 31, Apr = 30, May = 31, Jun = 30, Jul = 31, Aug =
31, Sep = 30, Oct = 31, Nov = 30,Dec = 31 )
>>> monthdays
{'May': 31, 'Aug': 31, 'Feb': 28, 'Mar': 31, 'Jan': 31, 'Jul': 31, 'Jun': 30, 'Sep': 30, 'Nov': 30,
'Dec': 31, 'Oct': 31, 'Apr': 30}
```

类型构造器对键值对的要求比字面值的键值对的要求更严格,键名 key 必须是一个标识符,而不能是表达式,例如:类似 d1 的字典不能使用类型构造器生成,因为整数不能作为 key。

【例 3-4-27】 使用类型构造器构造字典示例。

```
>>> weekday = dict(1 = 'MON',2 = 'TUE',3 = 'WED',4 = 'THU',5 = 'FRI',6 = 'SAT',0 = 'SUN')
SyntaxError: keyword can't be an expression
>>> weekday = dict(a1 = 'MON',a2 = 'TUE',a3 = 'WED',a4 = 'THU',a5 = 'FRI',a6 = 'SAT',a0 = 'SUN')
>>> weekday
{'a3': 'WED', 'a2': 'TUE', 'a1': 'MON', 'a0': 'SUN', 'a6': 'SAT', 'a5': 'FRI', 'a4': 'THU'}
```

2) 字典元素的访问

字典元素的访问方式是通过键访问相关联的值,访问形式为:<字典对象>[<键>]。

例如 monthdays["Jan"],可访问值 31。如果没有找到指定的键,则解释器会引起异常。

【例 3-4-28】 字典元素的访问。

```
>>> monthdays["Jan"]
31
>>> monthdays["Jau"]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    monthdays["Jau"]
KeyError: 'Jau'
```

3) 字典的基本运算

• 字典是可修改的。

【例 3-4-29】 monthdays["Jan"]=30,可把 Jan 的值由 31 改为 30。

```
>>> monthdays["Jan"] = 30
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul': 31,
'Mar': 31, 'Sep': 30, 'Nov': 30}
```

• 字典是可添加元素的。

【例 3-4-30】 monthdays["test"]=30 可添加一个新键值对。

```
>>> monthdays["test"] = 30
```

```
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul': 31,
'test': 30, 'Mar': 31, 'Sep': 30, 'Nov': 30}
```


- 字典是可删除元素的。

【例 3-4-31】 del monthdays["test"]可删除字典条目。

```
>>> del monthdays["test"]
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul': 31,
'Mar': 31, 'Sep': 30, 'Nov': 30}
```

字典不属序列对象,所以不能进行连接和相乘操作。字典是没有顺序的。

4) 字典对象的方法

和列表一样,字典也提供了对象方法来对字典进行操作。假设 d 为字典对象,字典对象的常用方法如表 3-4-6 所示。

表 3-4-6 字典对象的常用方法

方　法	描　述
d. keys()	返回字典 d 中所有键的列表,类型为 dict_keys
d. values()	返回字典 d 中值的列表,类型为 dict_values
d. items()	返回字典 d 中由键和相应值组成的元组的列表,类型为 dict_items
d. clear()	删除字典 d 的所有条目
d. copy()	返回字典 d 的浅复制拷贝,不复制嵌入结构
d. update(x)	将字典 x 中的键值加入到字典 d
d. pop(k)	删除键 k 的键值对,返回 k 所对应的值
d. get(k[,y])	返回键 k 对应的值,若未找到该键返回 none,若提供 y,则未找到 k 时返回 y

【例 3-4-32】 字典方法示例。

```
>>> monthdays.keys()          # 显示字典 monthdays 的键值序列
dict_keys(['Apr', 'Dec', 'May', 'Feb', 'Aug', 'Oct', 'Jan', 'Jun', 'Jul', 'Mar', 'Sep', 'Nov'])
>>> monthdays.values()        # 显示字典 monthdays 的值序列
dict_values([30, 31, 31, 28, 31, 31, 30, 30, 31, 31, 30, 30])
```

dict_keys 和 dict_values 也是一个迭代器对象,可以通过迭代方式访问其中的元素,例如:

```
>>> for i in monthdays.keys():
    print(i,end = " ")
Apr Dec May Feb Aug Oct Jan Jun Jul Mar Sep Nov
>>> monthdays.items()         # 显示字典 monthdays 的键值对序列
dict_items([('Apr', 30), ('Jul', 31), ('Jun', 30), ('Oct', 31), ('Mar', 31), ('Jan', 30), ('May', 31),
('Nov', 30), ('Dec', 31), ('Aug', 31), ('Sep', 30), ('Feb', 28)])

>> x = {'a1':21,'a2':34}      # 创建一个新的字典 x
>>> x
{'a2': 34, 'a1': 21}
>>> monthdays.update(x)       # 将字典 x 的键值对追加到字典 monthdays 中
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30, 'May': 31, 'Nov': 30, 'Dec': 31,
'a2': 34, 'a1': 21, 'Aug': 31, 'Sep': 30, 'Feb': 28}
```

```
>>> monthdays.pop('a1')        # 删除键为'a1'的键值对
21
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30, 'May': 31, 'Nov': 30, 'Dec': 31,
'a2': 34, 'Aug': 31, 'Sep': 30, 'Feb': 28}
>>> monthdays.get('a2')        # 获取键'a2'对应的值
34
>>> monthdays.get('a1','not found')   # 获取键'a1'对应的值,没有找到则返回'not found'
'not found'
```

5）应用

【例 3-4-33】 建立一个字典对象,能够通过数字 1～12 表示月份,查阅对应的英文月份的缩写。

创建一个 key 为序号,value 为英文月份缩写的集合。

```
>>> monthname = {1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May', 6:'Jun', 7:'Jul', 8:'Aug', 9:'Sep',10:
'Oct', 11:'Nov',12:'Dec' }
>>> monthname
{1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct',
11: 'Nov', 12: 'Dec'}
```

按 key 值查询对应英文月份的缩写。

```
>>> monthname [1]
'Jan'
```

输出所有 12 个月的英文月份的缩写。

```
>>> for i in monthname.keys():
    print(monthname [i],end = ' ')

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

【例 3-4-34】 建立 9×9 乘法表,可以根据两个乘数,查阅字典得到乘积。

以 3 的乘法为例:

```
>>> d1 = {(3,1):3,(3,2):6,(3,3):9,(3,4):12,(3,5):15,(3,6):18,(3,7):21,(3,8):24,(3,9):27}
>>> d1
{(3, 8): 24, (3, 2): 6, (3, 9): 27, (3, 3): 9, (3, 6): 18, (3, 7): 21, (3, 1): 3, (3, 4): 12,
(3, 5): 15}
>>> d1[(3,9)]
27
```

【例 3-4-35】 成绩排序。

已有 5 位同学的姓名和成绩,按成绩从高到低列出同学姓名,假设成绩没有重复值。

方法一:按<成绩:姓名>建立字典,从字典获取由成绩组成的列表,从高到低排序后,根据列表中的成绩,逐个从字典中查找对应的姓名,写出另一个列表。得到的新列表中的姓名就是按成绩排序的。

```
>>> scores = {85:"李鸣",74:"黄辉",92:"张檬",88:"于静颂",63:"钱多多"}
>>> scores
```

```
{88: '于静颂', 74: '黄辉', 92: '张檬', 85: '李鸣', 63: '钱多多'}
>>> L1 = list(scores.keys())
>>> L1
[88, 74, 92, 85, 63]
>>> L1.sort(reverse = True)
>>> L1
[92, 88, 85, 74, 63]

>>> L2 = []
>>> L2.append(scores[L1[0]])
>>> L2.append(scores[L1[1]])
>>> L2.append(scores[L1[2]])
>>> L2.append(scores[L1[3]])
>>> L2.append(scores[L1[4]])
>>> L2
['张檬', '于静颂', '李鸣', '黄辉', '钱多多']
```

得到 L2 列表的过程在学习了循环结构后可改为

```
>>> L2 = []
>>> for i in range(0,len(L1)):
        L2.append(scores[L1[i]])
>>> L2
```

* 方法二：直接利用嵌套列表的 sort 方法的 key 参数，写一个 lambda 函数，对每一个列表成员 x 返回第二项 x[1]，即按第二项排序。reverse 为 true，表示倒序从高到低。

```
>>> Ls = [["李鸣",85],["黄辉",74],["张檬",92],["于静颂",88],["钱多多",63]]
>>> Ls.sort(key = lambda x:x[1], reverse = True)
>>> Ls
[['张檬', 92], ['于静颂', 88], ['李鸣', 85], ['黄辉', 74], ['钱多多', 63]]
>>> for i in range(0,len(Ls)):
    print(Ls[i][0],end = ' ')

张檬 于静颂 李鸣 黄辉 钱多多
```

3.5 本章小结

本章所介绍的主要内容是数值数据对象、文本数据对象和批量数据对象的常量表示和对象创建的方法，以及作用在这些数据对象上的基本操作：如何访问数据对象、数据对象支持的运算以及数据对象提供的方法。

本章的内容是学好 Python 语言的基础，重要的语法知识包括：

(1) 数据都是属于一定类型的，数据类型是一组数据及在这组数据上的运算，它规定了这一类数据：①存储结构；②存储机制，即各种数据类型的编码方式；③运算和操作。

(2) Python 以类的方式管理数据，Python 的内置类型主要区分为简单类型和容器类型，简单类型主要是数值型数据，包括整型数据、浮点型数据、布尔类型数据和其他语言不多见的复数数据。容器类型可以应用于一次处理多个对象的场合，包括字符串 str、元组

tuple、列表 list、集合类型 set、字典类型 dict。

(3) 程序中数据有两种表示方式：常量和变量。常量是数据的文字量，是数据的“书写形式”。变量描述的是存储空间的概念，将数据存储在内存中，内存空间就是可操作的变量，用一个名称来引用内存空间，这个名称称为变量名。变量的值是可以变化的。Python 语言使用“动态类型”技术，变量使用前不需要声明数据类型即可使用，然后根据变量存放的数据不同，决定其数据类型。

(4) 标识符用来标识一个对象，Python 中的标识符由大小写英文字母、数字、下划线组成，以英文字母、下划线为首字符，也就是说不能以数字开头，长度任意，大小写敏感。标识符不能与 Python 关键字同名。

(5) Python 的表达式可以由常量、变量、运算符、函数、运算符按照一定的规则构造，描述计算过程。最简单的表达式可以是一个常量或一个变量。

(6) 数值运算主要包括算术运算、关系运算和逻辑运算。算术运算有＋、－、*、/、//、%；关系运算有＜、＜＝、＞、＞＝、＝＝、!＝；逻辑运算有 and、or、not。

(7) 影响表达式计算顺序的因素包括：运算符的优先级、运算符的结合方式和括号。数值运算的优先级是先算术运算，再关系运算，最后是逻辑运算。算术运算同样遵循先乘除后加减。逻辑运算的优先级是先“非”再“与”最后“或”。括号的优先级最高。

(8) 只有同类型的数据对象才能进行运算，混合类型的数据进行运算时要进行类型转换。系统有自动转换的机制，自动转换的基本原则是将表示数值范围小的数据类型的值转换到表示数值范围大的数据类型的值。强制转换机制是指程序员可以使用 Python 语言提供各种类型的转换函数在表达式中明确混合运算中数据的转换类型。

(9) Python 的系统函数扩充了 Python 的计算能力，系统函数由 Python 标准库中的模块提供。标准库中的模块，又分成内置模块和非内置模块，内置模块__builtin__中的函数和变量可以直接使用，非内置模块要通过 import 命令先导入再使用。

(10) 计算机中表示文本的最基本的单位是字符，包括可打印字符和不可打印的控制字符。可打印的字符直接由键盘输入，不可打印的字符以转义字符表示，Python 中的转义字符以'\'为前缀。

(11) 字符串常量以一对双引号或单引号表示，字符串类型支持的运算有＋和 *，实现连接和复制。可以通过下标访问字符串中的一个字符或一个子串，也称为索引方式。但不能通过下标访问方式去改变字符串的内容。

(12) Python 可支持批量数据存储和操作，其中有序的数据集合体，称为序列，包括字符串、元组和列表，序列可以通过索引或下标来访问其数据成员，序列的通用操作包括索引、联接、复制、检测等；无序的数据集合体包括集合、字典等，无序的数据集合体，不支持索引操作。

(13) 批量数据对象的创建可以通过字面形式，给对象赋常量值，也可以通过类型构造器创建。例如创建一个空的元组对象，可以直接将一个空的元组赋给元组对象：t＝()；也可以使用无参的元组类型构造器创建：t＝tuple()。

(14) 每一种批量数据对象都提供了丰富的方法，以支持对批量数据对象的各种操作，方法的调用形式为：＜对象名＞. 方法名(＜参数＞)。

3.6 习题与思考

3-1 请指出下面合法的 Python 标识符是________。

A. Day　B. e10　C. 2n　D. a[10]
E. False　F. aAbB　G. a+b　H. _ifdef
I. day_of_year

3-2 以下是出现在程序中的数值常量,正确的是________。

A. 38499L　B. .314e1　C. e5　D. 1e2.5
E. 0o378　F. 0xabc　G. 0b1010　H. true
I. 5-6.5j　J. 78.90

3-3 以下是出现在程序中的文本常量,正确的是________。

A. """　B. 'ab'　C. '*'　D. '"ab"'
E. ""a""　F. '\456'　G. '\"'　H. '\xah'
I. "a+b"

3-4 写出执行完下面的数值表达式语句后,变量 $a \sim k$ 的值。

```
>>> a = 5
>>> b = 2
>>> a * = b
>>> b += a
>>> a, b = b, a
>>> c = 6
>>> d = c % 2 + (c + 1) % 2
>>> e = 2.5
>>> f = 3.5
>>> g = (a + b) % 3 + int(f)//int(e)
>>> h = float(a + b) % 3 + int(f)//int(e)
>>> i = (a + b)/3 + f % e
>>> j = a < b and c < d
>>> k = not j and True
```

3-5 已知 s="Happy Birthday",写出下面输出语句 print 的输出结果。

(1) print(len('\\n\n\456'))

(2) print(('hello '+'world\n') * 3)

(3) print(s[0]+s[11])

(4) print(s[6:11])

(5) print(s[:5]+s[11:])

3-6 已知列表 L1 和 L2,由 L1 和 L2 构造 L3,并回答问题。

```
>>> L1 = [1,2,3,4,5]
>>> L2 = ["one","two","three","four","five"]
>>> L3 = [[L1[1],L2[1]],[L1[2],L2[2]],[L1[3],L2[3]]]
```

(1) L3 的值是什么？

(2) L3[1,1]的值是什么？

(3) 执行 L4=L3.pop(2)后，列表 L3 和 L4 的值是什么？

(4) 再执行 L3.extend(L4)，列表 L3 的值是什么？

3-7 集合 a、b 中存放着两组文件名的集合，两个集合中有相同的文件也有不同的文件，请写出实现下面功能的表达式。

a={"3-1.py","3-5.py","3-6.py","3-8.py","3-9.py"}

b={"3-1.py","3-2.py","3-6.py","3-7.py","3-8.py"}

(1) 求 a 中存在，b 中不存在的文件；

(2) 求 a 中存在的与 b 中相同的文件；

(3) 求两个文件夹中互不相同的文件；

(4) 求两个文件夹中总共包括的文件的个数。

3-8 下面定义字典 monthdays 的语句都正确吗？如果不正确，说说为什么？

(1) monthdays = dict(Jan=31, Feb=28, Mar=31, Apr=30, May=31, Jun=30, Jul=31, Aug=31, Sep=30, Oct=31, Nov=30,Dec=31)

(2) monthdays = dict('Jan'=31, 'Feb'=28, 'Mar'=31, 'Apr'=30, 'May'=31, 'Jun'=30, 'Jul'=31, 'Aug'=31, 'Sep'=30, 'Oct'=31, 'Nov'=30,'Dec'=31)

(3) monthdays = {Jan=31, Feb=28, Mar=31, Apr=30, May=31, Jun=30, Jul=31, Aug=31, Sep=30, Oct=31, Nov=30,Dec=31}

(4) monthdays = {Jan:31, Feb:28, Mar:31, Apr:30, May:31, Jun:30, Jul:31, Aug:31, Sep:30, Oct:31, Nov:30,Dec:31}

(5) monthdays = {'Jan':31, 'Feb':28, 'Mar':31, 'Apr':30, 'May':31, 'Jun':30, 'Jul':31, 'Aug':31, 'Sep':30, 'Oct':31, 'Nov':30,'Dec':31}

3-9 思考在下面举出的应用环境中适合的数据类型，试在 Python Shell 中举实例表示。

(1) 100 以内的素数。

(2) 1～20 的数字的阶乘。

(3) 费波纳茨数列。

(4) 班级考试成绩单。

(5) 自定义的英汉字典。

3.7 实验 数据的表示和操作

3.7.1 实验目标

(1) 理解数据类型的概念和数据的文字量表示。

(2) 掌握数值类型和文本类型的基本操作。

(3) 理解变量的存储概念。

(4) 掌握列表和元组的概念和基本操作。

3.7.2 实验范例

本章的实验使用 Python 的交互模式,像使用计算器一样使用 Python。交互模式可以在 Python(commands line)环境或 Python Shell 环境下使用。

(1) Python(commands line)。

请从开始菜单中选择打开 Python33→Python(commands line),等待提示符＞＞＞。在窗口的第一行显示 Python 的版本号,如图 3-7-1 所示。

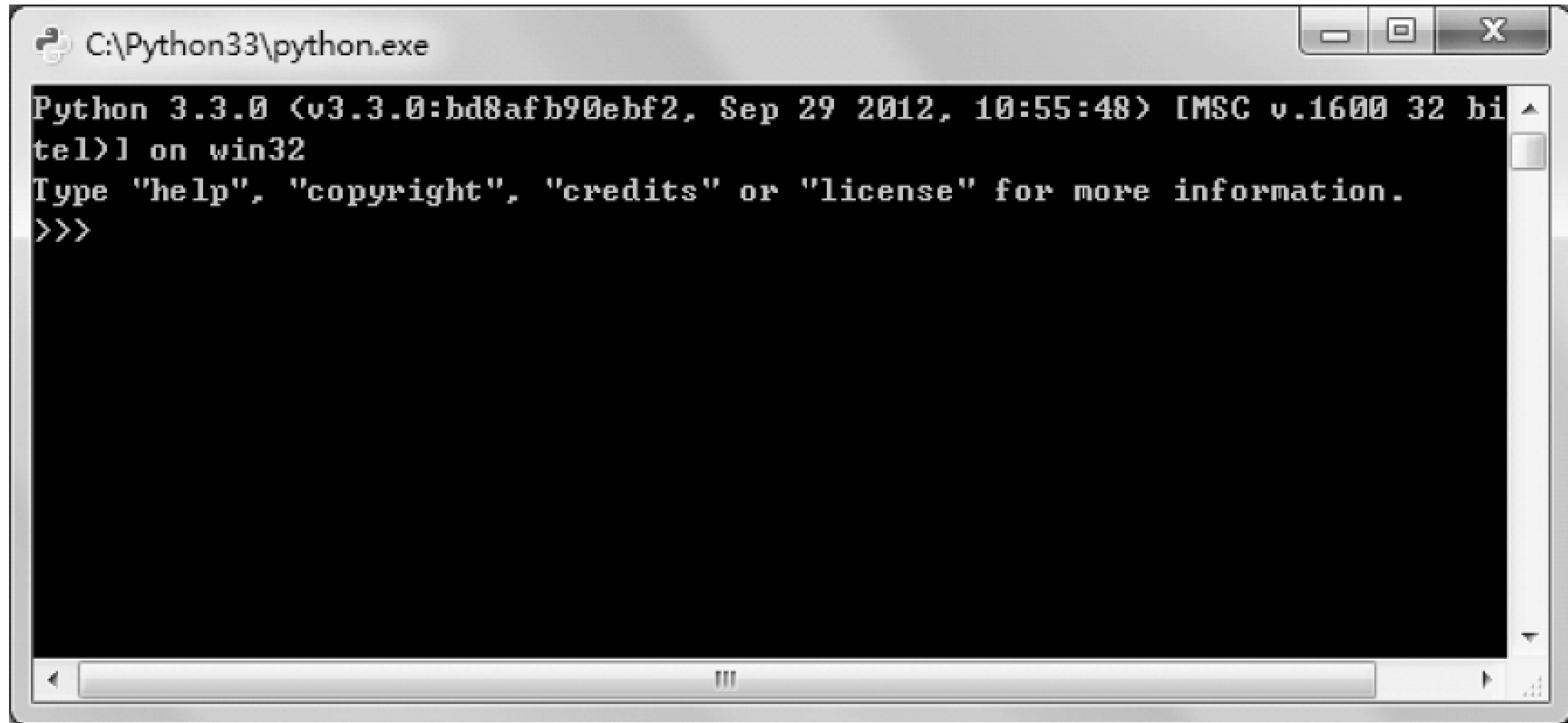


图 3-7-1　Python(commands line)

(2) Python Shell。

请从开始菜单中选择打开 Python33→IDLE(Python GUI),如图 3-7-2 所示。

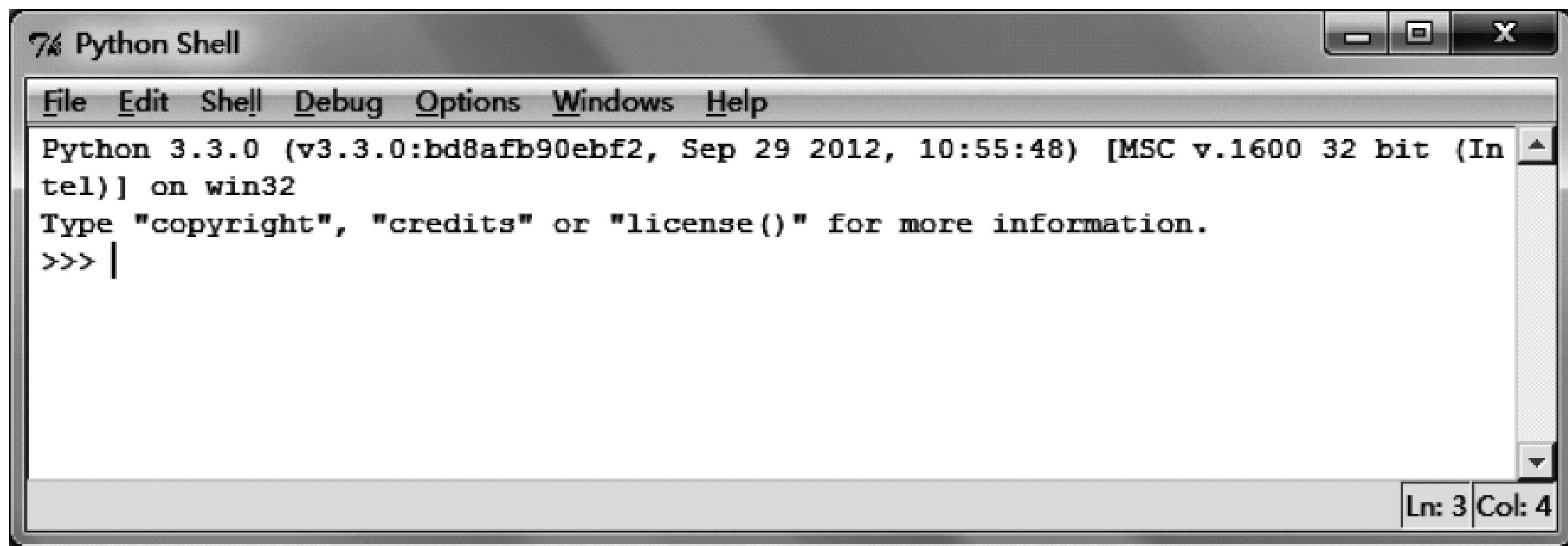


图 3-7-2　Python Shell

这两种方式都支持 Python 交互模式,区别在于 Python(commands line)是 DOS 控制台模式,不支持鼠标操作,Python Shell 是窗口模式,支持鼠标操作,也支持剪贴板操作。

例如在交互模式中,要想重复执行前面已执行过的命令,或想获得前面已执行过的命令修改得到新的命令,Python(commands line)中使用光标键↑,上翻到所需命令。Python Shell 中可以用复制粘贴的方法,更快捷的操作是在已完成的命令行任意位置单击将光标插入文本后按 Enter 键,该行文本会自动复制到当前等待输入的命令行提示符的后面,可进行修改后或直接按 Enter 键再次执行。

1. 认识基本数据的类型、表示和运算

(1) 直接输入以下表达式并查看结果。

```
23 + 3、 23 > 3、 '23' + '3'、  23/3、  23//3、   23 % 3、   23 * * 3
>>> 23 + 3
26
>>> 23 > 3
True
>>> '23' + '3'
'233'
>>> 23/3
7.666666666666667
>>> 23//3
7
>>> 23 % 3
2
>>> 23 * * 3
12167
```

注意：命令行提示符后不要插入空格，否则会引起系统错误。

```
>>>   23 % 3
SyntaxError: unexpected indent
```

(2) 直接输入以下表达式并查看结果。

```
23 + 24.5、23 + '3'、23 + int('3')、'hello ' + str("123")、int(23/3)、round(23/3,2)、round(23/3)、
0 < 23 < 100
```

不同的数据类型进行运算时，会进行类型的转换，整型数据和浮点数据相遇，整型数据转化为浮点类型。

```
>>> 23 + 24.5
47.5
```

当自动类型转化不成功或出现系统错误，例如整型与字符串类型相加出错时。

```
>>> 23 + '3'
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    23 + '3'
TypeError: unsupported operand type(s) for + : 'int' and 'str'
```

但可以通过类型转化函数，显示完成数据类型转换后计算。

```
>>> 23 + int('3')
26
>>> 'hello ' + str(123)
'hello 123'
```

int 函数还可以完成对浮点数取整的功能。

```
>>> int(23/3)
7
```

round 函数的功能更为灵活，可以按指定位取整，四舍五入。第二个参数指定取整位置，n 表示小数点后 n 位，缺省表示没有小数点。

```
>>> round(23/3,2)
7.67
>>> round(23/3)
8
```

Python 支持下面连写的比较方式,但与其他高级语言迥异。

```
>>> 0 < 23 < 100
True
```

与其他高级语言相同的写法应为

```
>>> (23 > 0) and (23 < 100)
True
```

建议使用逻辑运算的方式表示多个关系表达式之间的关系。

```
>>> 'hello ' + str("123")
'hello 123'
```

(3) 直接输入变量赋值语句并接着显示该变量值或类型。

输入:$a=23.5$,再输入:a 显示该变量值,最后输入:type(a)显示变量类型。

```
>>> a = 23.5
>>> a
23.5
>>> type(a)
<class 'float'>
```

输入:$b=a>0$,再输入:b 显示该变量值,最后输入:type(b)显示变量类型。

```
>>> b = a > 0
>>> b
True
>>> type(b)
<class 'bool'>
```

输入:$c=$None,再输入:c 显示该变量值,最后输入:type(c)显示变量类型。

```
>>> c = None
>>> c
>>> type(c)
<class 'NoneType'>
```

注:None 表示空类型。

输入:$d=$'23'+'3',再输入:d 显示该变量值,最后输入:type(d)显示变量类型,输入:len(d)显示变量长度。

```
>>> d = '23' + '3'
>>> d
'233'
>>> type(d)
<class 'str'>
```

2. 数学模块库函数的使用

使用 math 模块的数学函数。导入数学库 math。然后输入以下表达式理解 math 中函数的使用。

math.sqrt(2 * 2 + 3 * 3)、math.log10(100)、math.exp(2)、math.e、math.pow(2.5,2)、math.floor(2.5)、math.ceil(2.5)、math.fmod(4,3)、math.fabs(-23.56)

导入数学库。

```
>>> import math
```

使用 math 前缀访问 sqrt 函数求平方根。

```
>>> math.sqrt(2 * 2 + 3 * 3)
3.605551275463989
>>> math.log10(100)
2.0
```

另一种导入数学库的方式，访问 pow 函数求 x 的 y 次方。

```
>>> from math import *
>>> pow(2.5,2)
6.25
>>> pi
3.141592653589793
>>> e
2.718281828459045
>>> math.floor(2.5)
2
>>> ceil(2.5)
3
>>> fabs(-23.56)
23.56
```

3. 变量的表示和操作

(1) 输入平面的两个点的坐标(1.0,2.1),(5.2,10.33),计算两点之间的距离。计算两点之间 $p_1(x_1,y_1)$,$p_2(x_2,y_2)$距离的公式为：$d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$。

```
>>> from math import *
>>> x1,y1 = 1.0,2.1
>>> x2,y2 = 5.2,10.33
>>> d = sqrt(pow(x2 - x1,2) + pow(y2 - y1,2))
>>> print(d)
9.239745667495399
```

增加 x_2 和 y_2 的值，再计算。

```
>>> i = 5
>>> x2,y2 = x2 + i,y2 + i
>>> d = sqrt(pow(x2 - x1,2) + pow(y2 - y1,2))
>>> print(d)
16.114369364017943
```

(2) 求任意三位整数的逆序数。

求一个三位数的逆序数的思路是将构成三位数的三个数字取出，重新按权位组合。取

位可以通过求余和整除运算完成。

```
>>> a = 123
>>> d1,d2,d3 = a//100,a % 100//10,a % 10
>>> d1,d2,d3
(1, 2, 3)
>>> b = d3 * 100 + d2 * 10 + d1
>>> b
321
```

4. 文本对象的表示和操作

假设执行了以下语句:

```
>>> s1 = 'programming'
>>> s2 = 'language'
```

直接输入下面的表达式,观察计算结果,理解表达式的含义。

s1[1]、s1[:4]、s1[0] + s2[1:3]、s1.capitalize() + ' ' + s2.upper()、s1.count('r') + s1.find('r') + s1.rfind('r')、s3 = s2.join('-- ')、s4 = '- '.join(s2)、L1 = s4.split()、3 * (s2[:2] + ' ')、"Python" + s2.rjust(10)。

(1) 取下标为 1 的字符串字符。

```
>>> s1[1]
'r'
```

(2) 取下标从开始到 3 的子串。

```
>>> s1[:4]
'prog'
```

(3) 连接 $s1$ 串下标为 0 的字符和 $s2$ 串下标从 1 到 2 的子串。

```
>>> s1[0] + s2[1:3]
'pan'
```

(4) 连接首字符大写后的 $s1$ 串和全部大写的 $s2$ 串。

```
>>> s1.capitalize() + ' ' + s2.upper()
'Programming LANGUAGE'
```

(5) 计算 $s1$ 串中的字符 r 的个数,从字符 r 在 $s1$ 串的左边和右边第一次出现的位置。(位置都从左边开始计数,第一个位置为 0),并累和。

```
>>> s1.count('r') + s1.find('r') + s1.rfind('r')
7
```

(6) 将字符串 $s2$ 作为分隔符加入序列参数--的每个字符之间。

```
>>> s3 = s2.join('-- ')
>>> s3
'- language - '
```

(7) 将字符串-作为分隔符加入参数 $s2$ 串的每个字符之间。

```
>>> s4 = '-'.join(s2)
>>> s4
```

(8) 以参数字符-为分隔符，将字符串 $s4$ 分离到一个列表中。

```
'l-a-n-g-u-a-g-e'
>>> L1 = s4.split('-')
>>> L1
['l', 'a', 'n', 'g', 'u', 'a', 'g', 'e']
```

(9) 取串 $s2$ 开始两个字符连接一个空格后复制三次。

```
>>> 3 * (s2[:2] + ' ')
'la la la '
```

5. 序列的表示和操作

(1) 输入：$t1$='001001', 'Li Si', 'men', 18，再输入：$t1$ 显示该变量值，输入：$t1$[0]和 $t1$[1]显示部分数据，最后输入：type($t1$)显示变量类型，输入：len($t1$)显示长度。

```
>>> t1 = '001001', 'Li Si', 'men', 18
>>> t1
('001001', 'Li Si', 'men', 18)
>>> t1[0]
'001001'
>>> t1[1]
'Li Si'
>>> type(t1)
<class 'tuple'>
>>> len(t1)
4
```

(2) 输入：$t2$=['001001','Li Si','men',18]，再输入：$t2$ 显示该变量值，输入：$t2$[0]和 $t2$[1]显示部分数据，最后输入：type($t2$)显示变量类型，输入：'men' in $t2$ 测试成员。

```
>>> t2 = ['001001','Li Si','men',18]
>>> t2
['001001', 'Li Si', 'men', 18]
>>> t2[0]
'001001'
>>> t2[1]
'Li Si'
>>> type(t2)
<class 'list'>
>>> 'men' in t2
True
```

(3) 输入：$t2$[3]+=1，再输入：$t2$ 查看该变量值。输入：$t1$[3]+=1 显示出错信息。

```
>>> t2[3] += 1
>>> t2
['001001', 'Li Si', 'men', 19]
```

```
>>> t1[3] += 1
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    t1[3] += 1
TypeError: 'tuple' object does not support item assignment
```

这说明列表的元素可以改变,而元组的元素不可以改变。

(4) 输入:$t2$+=['021-65789293'],再输入:$t2$,查看该变量值。输入:$t2$[0:1]=[],再输入:$t2$,查看该变量值。

```
>>> t2 += ['021 - 65789293']
>>> t2
['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t2[0:1] = []
>>> t2
['Li Si', 'men', 19, '021 - 65789293']
```

(5) 将 $t2$ 的内容复制一个副本 $t3$,对 $t3$ 进行增删修改。

直接使用赋值运算得到的 $t3$,与 $t2$ 是指向同一个列表对象的,对 $t2$ 和 $t3$ 的操作实质是使用不同的名称对一个对象操作。

```
>>> t2 = ['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t2
['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t3 = t2
>>> t3[3] = 20
>>> t2
['001001', 'Li Si', 'men', 20, '021 - 65789293']
>>> id(t2),id(t3)
(33775328, 33775328)
```

要得到一个对象副本,可使用列表的方法 copy,复制后,两个列表的内容是相等的,但不是一个对象,注意==运算和 is 运算的区别。

```
>>> t3 = t2.copy()
>>> t3
['001001', 'Li Si', 'men', 20, '021 - 65789293']
>>> t3 == t2
True
>>> t3 is t2
False
>>> id(t2),id(t3)
(33775328, 34958072)
```

也可以从一个空列表开始,将 $t2$ 的内容加入空列表中。设置一个空列表对象是必须的,因为之前 $t3$ 并没有指定一个确定的数据类型。

```
>>> t3 = []
>>> t3.extend(t2)
>>> t3
['001001', 'Li Si', 'men', 19, '021 - 65789293']
```

通过 append 方法可以在列表的尾部追加一个列表成员，例如增加一个身高的信息。

```
>>> t3.append(1.78)
>>> t3
['001001', 'Li Si', 'men', 19, '021 - 65789293', 1.78]
```

注意 append 与 extend 的区别。如果调用 $t3$.append($t2$)，它是将列表 $t2$ 作为一个列表成员加入的，而不是将列表 $t2$ 的每一个成员分别加入的。

```
>>> t4 = []
>>> t4.append(t2)
>>> t4
[['001001', 'Li Si', 'men', 19, '021 - 65789293']]
>>> len(t4)
1
```

$t4$ 中只嵌套了一个列表成员。

insert 方法支持在指定位置增加列表成员，例如在年龄的后面插入身高信息。

```
>>> t3.insert(4,1.78)
>>> t3
['001001', 'Li Si', 'men', 19, 1.78, '021 - 65789293', 1.78]
```

remove 方法可用于删除第一个指定值，pop 可以删除并返回指定位置的列表成员，例如将多余的身高成员删除。

```
>>> t3.pop(6)
1.78
>>> t3
['001001', 'Li Si', 'men', 19, 1.78, '021 - 65789293']
```

3.7.3 实验内容

(1) 查阅 Python 3.3.0 自带的帮助文件 Python 330.chm，文件存放位置为 Python33\Doc 文件夹，了解学习 Python 3.3.0 提供的内置函数(Built-in Functions)，写出其中 5 个你学会的函数的使用示例。

(2) 假设执行了以下语句。

```
>>> x = 384
>>> a,b = 2.56769, 2.56789
>>> s1 = "She is the best student in her class"
>>> s2 = 'he'
```

写出下面条件判断语句：

① 判断 x 是否是奇数。

② 判断 x 是否能被 3 和 5 整除。

③ 判断 x 是否能被 3 或 5 整除。

④ 判断 b 与 a 的差值不超过 0.0001。

⑤ 判断 $s2$ 是 $s1$ 的子串。

⑥ 判断 s2 在 s1 中出现的次数超过 2 次。

(3) 假设执行了以下语句。

```
>>> s1 = 'programming'
>>> s2 = 'language'
```

利用 s1、s2 和字符串操作,写出能产生下列结果的表达式。

① 'program'

② 'ProLan'

③ 'am am am'

④ 'programming   language'

⑤ 'progr@mming l@ngu@ge'

(4) 假设执行了以下语句。

```
>>> s1 = [0,1,2,3,4,5,6]
>>> s2 = ['SUN','MON','TUE','WED','THU','FRI','SAT']
```

利用 s1、s2 和列表操作,创建下列结果的序列对象(可分次完成)。

```
s3: 'SUN|MON|TUE|WED|THU|FRI|SAT'
s4: [3, 4, 3, 4, 3, 4]
s5: [[0, 'SUN'], [1, 'MON'], [2, 'TUE'], [3, 'WED'], [4, 'THU'], [5, 'FRI'], [6, 'SAT']]
```

(5) 程序设计:使用圆柱的体积公式计算已知半径和高的圆柱的体积。

(6) 程序设计:输入连续 5 天的气温,求平均气温。

(7) 程序设计:使用查表法完成成绩类别的转换。

① 求任意一个分数(5 分制)对应等级。

1:A,2:B,3:C,4:D,5:E。

② 求任意一个分数(百分制)对应等级。

90～100:A,80～89:B,70～79:C,60～69:D,0～59:E。

第4章 基本控制结构的程序设计

"结构程序设计"概念被称为软件发展中的第三个里程碑(第一、第二个里程碑是子程序和高级语言),是由著名的荷兰计算机科学家埃德斯加·狄克斯特拉(Edsgar Wybe Dijkstra)提出的。

早在1965年召开的IFIP会议上,狄克斯特拉就提出"GOTO语句可以从高级语言中取消","一个程序的质量与程序中所含的GOTO语句的数量成反比"。在1966年,C Bohm和G Jacopini证明了程序设计语言中,只要有顺序、选择和循环三种形式的控制结构,就足以表示出其他各式各样的程序结构。1968年3月,ACM通信(Communications of ACM)登出了狄克斯特拉的那封影响深远的信《GOTO语句看来是有害的》(*GOTO Statement Considered Harmful*),在信中他根据自己编程的实际经验和大量观察,得出以下结论:一个程序的易读性和易理解性同其中所包含的无条件转移控制的个数成反比关系,也就是说,转向语句的个数越多,程序就越难读、难懂。因此他认为"GOTO是有害的",从而启发了结构化程序设计的思想。1972年,他与当时在爱尔兰昆士大学任教的英国计算机科学家、1980年图灵奖获得者霍尔(C. A. R. Hoare)合著了《结构程序设计》一书(*Structured Programming*,Academic Pr.),进一步发展与完善了这一思想,并且提出了另一个著名的论断:程序测试只能用来证明有错,决不能证明无错!(Program testing can be used to show the presence of bugs,but never to show their absence!)有关GOTO语句的争论,直到1974年克努特发表文章《带有GOTO语句的结构化程序设计》之后才平息下来。他主张在语言控制划分中仍然保留GOTO语句,在功能方面不加限制,但限制其使用范围。结构化程序允许有GOTO语句,但它只能在本程序块内使用,不允许从一个结构转移到另一个结构。

在与癌症进行了多年的斗争之后,伟大的荷兰计算机科学家埃德斯加·狄克斯特拉已经于2002年8月6日在荷兰自己的家中与世长辞,终年72岁。

"结构程序设计"的主要观点是采用自顶向下、逐步求精的程序设计方法;使用三种基本控制结构构造程序,任何程序都可由顺序、选择、重复三种基本控制结构构造;其实质是控制编程中的复杂性,该方法的要点是:

(1) 没有GOTO语句;

(2) 一个入口,一个出口;

(3) 自顶向下、逐步求精的分解;

(4) 主程序员组。

其中(1)、(2)用于解决程序结构规范化问题;(3)用于解决将大划小,将难化简的求解方法问题;(4)用于解决软件开发的人员组织结构问题。

尽管计算机语言有万千种,但它们都无一例外地支持这三种结构。

4.1 用 Python 实现顺序结构程序

顺序结构是最基本的程序结构,它最符合我们的书写习惯,从上至下,逐行执行。它是如此简单,以至于很多资料不会专门把它叫做一种结构。图 4-1-1 直观地解释了顺序结构的代码执行过程。

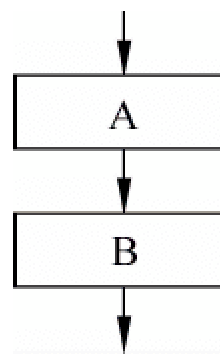


图 4-1-1 顺序执行的流程图

程序从入口开始,自顶向下,依次执行 A 代码块,B 代码块,A、B 的内容可以继续是一个顺序结构,又或者是分支、选择结构。这也叫做程序的嵌套,正是因为嵌套的存在,使得程序的执行顺序,可以灵活变化,从而实现不同的功能。

【例 4-1-1】 温度转换。

问题分析和算法可以参考第 2 章的例 2-2-1,温度华氏和摄氏温度两者的对应关系是 $F=(9/5)C+32$。

在 Python 的 IDLE 环境下创建一个名为 temperature.py 的程序,代码如下所示:

```
celsius = int(input('请输入一个摄氏温度: '))
fahrenheit = (9/5) * celsius + 32
print('华氏温度: ')
print(fahrenheit)
```

执行结果示例:

```
>>>
请输入一个摄氏温度: 33
华氏温度:
91.4
>>>
```

该示例的程序结构就是典型的顺序结构,程序自上而下执行,结构清晰,但不够灵活。例如,在正常的情况下,绝对零度只能无限接近,所以温度不低于 −273.15 摄氏度或者 −459.67 华氏度,如果能对这种情况进行预判,可以提高输错的概率。顺序结构显然无法实现这个功能。

4.2 用 Python 实现分支结构程序

“分支结构”在很多参考资料中也被称为选择结构,它的出现一定程度上改变了顺序结构所带来的弊端。因为在很多情况下,可能更希望满足某种条件才会去执行某些特定的语句,而并不希望计算机不假思索、一如既往地去顺序执行。

例如:想向用户输出一个问候语句“早上好”或者“下午好”,而这个输出将依赖于现在的时间,那么时间就是一个条件,根据对这个条件的判断,决定程序的走向,即该输出哪条问候语句。

根据条件的特点，分支结构又可以分为简单分支、双分支，以及多分支等。

4.2.1 Python 简单分支

在这种分支结构中，通常是满足某种条件，就执行某些语句，如果没有满足条件，则不执行相应的语句。它的语法结构是：

```
if 条件:
    条件成立时执行的代码块
```

流程图如图 4-2-1 所示。

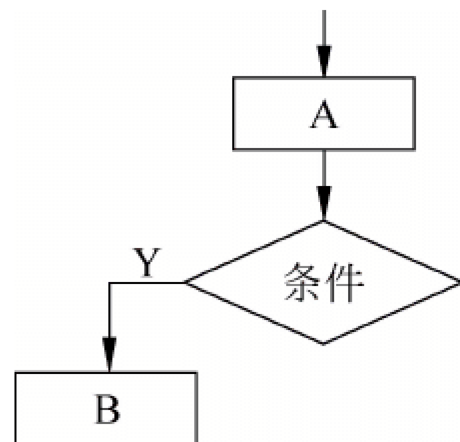


图 4-2-1 简单分支的流程图

在这个结构中，充当条件的往往是关系表达式或逻辑表达式，在条件后面不能丢失冒号(:)。

【例 4-2-1】 单分支两个数求最大值。

在 Python 的 IDLE 环境下创建一个名为 if_statement.py 的程序，代码如下所示：

```
numA = 3
numB = 4
if numA <= numB:
    print ("numB 是比较大的数")
```

以上程序的意思就是，如果变量 numA 的值小于等于 numB 的值，就输出“numB 是比较大的数”；在这种结构中如果变量 numA 的值不小于等于 numB 的值，程序则不予考虑，如果需要考虑的话，上述程序需要变为下面的语句：

```
numA = 3
numB = 4
if numA <= numB:
    print("numB 是比较大的数")
if numB <= numA:
    print("numA 是比较大的数")
```

运行结果如下：

```
>>>
numB 是比较大的数
>>>
```

4.2.2 Python 双分支

这种分支结构的执行过程是：满足某种条件，执行某些语句；否则，条件不满足的时候，就执行另一些语句。这种分支结构编程通常采用下列语法结构：

```
if 条件:
    满足条件时需要执行的语句
else:
    不满足条件时需要执行的语句
```

这种结构的流程图如图 4-2-2 所示。

这种结构只比上一种结构多了一个 else:,这个 else:之后的代码,就是不满足条件时需要执行的分支。有了这样的双分支结构,像上述求两个数最大值的问题,从逻辑上互相排斥的两个条件,仅仅使用一个双分支结构就可以取代上面的两个单分支结构。

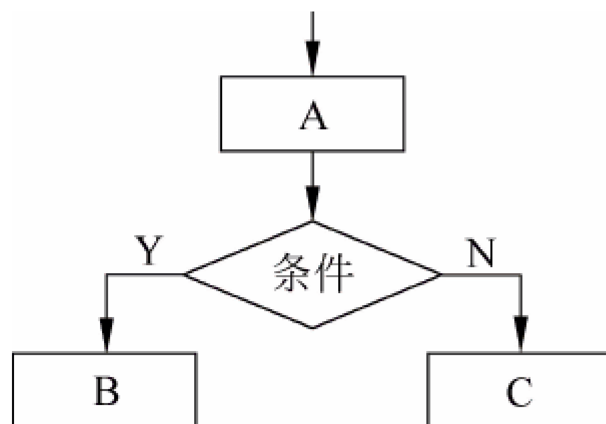


图 4-2-2 双分支执行的流程图

【例 4-2-2】 双分支求两个数的最大值。

在 Python 的 IDLE 环境下创建一个名为 if_else_statement. py 的程序,代码如下所示:

```
numA = 3
numB = 4
if numA < = numB:
    print("numB 是比较大的数")
else:
    print("numA 是比较大的数")
```

上述代码同样可以实现求两个数最大值的功能,但是逻辑思维上则更加清晰。

【例 4-2-3】 双分支改进华氏摄氏度和温度的转换。

在 Python 的 IDLE 环境下创建一个名为 if_else_temperature. py 的程序,代码如下所示:

```
d = input('请输入华氏温度:')
if float(d) <= - 459.67:
   print("输入错误")
else:
   print("摄氏温度是:")
   print(round((float(d) - 32)/1.8,2))
```

上述代码对例 4-2-1 进行了改进,可以确保输入华氏温度的时候,不会输入绝对零度。

思维拓展:if float(d)<= -459.67:是否可以写成 if d<=-459.67:? 答案是否定的,因为默认的 input 传递给变量的数值的数据类型都是文本类型,而文本类型的数据在执行比较的时候,是以 ASCII 顺序比较的,而不是数学意义上的比较。而例 4-2-2 中,为什么可以使用 if numA<=numB:对 numA 和 numB 比较,这是因为 numA=3 语句执行后,numA 的数据类型自动转换为整数。同理适用于 numB。所以可以直接使用 numA<=numB 进行比较。

4.2.3 Python 分支嵌套

不管是单分支还是双分支结构,都可以实现嵌套功能。此种嵌套可以将条件更加细分。

【例 4-2-4】 分支嵌套求三个数的最大值。

利用求两个数最大的思维方式,求三个数的最大值。在 Python 的 IDLE 环境下创建一个名为 if_else_nest. py 的程序,代码如下所示:

```
numA = 3
numB = 4
numC = 5
```

```
if numA <= numB:
    if numC < numB:
            print ("numB 是最大的数")
    else:
            print ("numC 是最大的数")
else:
    if numC < numA:
            print ("numA 是最大的数")
    else:
            print ("numC 是最大的数")
```

程序的运行结果示例：

```
>>>
numC 是比较大的数
>>>
```

以上程序的意思就是首先比较 numA 和 numB 的大小，如果 numB 大的话，则比较 numB 和 numC 的大小，如果 numA 大的话，则比较 numA 和 numC 的大小。需要注意的是 Python 的嵌套是依靠程序代码的缩进实现的，所以在书写代码的时候要注意 Python 的风格和代码习惯。

4.2.4 Python 多分支结构

有时候考虑问题往往不只有两个方面，可能会有很多方面，而且根据不同的条件，都需要对这很多个方面进行处理，这就需要用到多分支结构。这种结构与上述两种结构相比，最大的不同点在于，它的条件不再是单一的满足和不满足，而是可能有很多个条件，每个条件都可以构成一个分支，当然，这些条件必须互相排斥，因此，这种结构也被称为多分支结构。

```
if 条件 1:
    满足条件 1 需要执行的语句
elif 条件 2:
    满足条件 2 需要执行的语句
elif 条件 3:
    满足条件 3 需要执行的语句
⋮
elif 条件 N:
    满足条件 N 需要执行的语句
else
    以上 N 个条件都不满足的需要执行的语句
```

流程图如图 4-2-3 所示。

这种结构的特点主要体现在若干个 elif …：，它们构成了若干个条件的判断，要特别注意在这种语法结构中，elif 之间不存在空格，不能写成 el　if，而且每个 elif 后面都有一个冒号。

语句的缩进量要保持一致。在 Python 中没有 switch 和 case 语句，可通过多重 elif 来达到相同的效果。

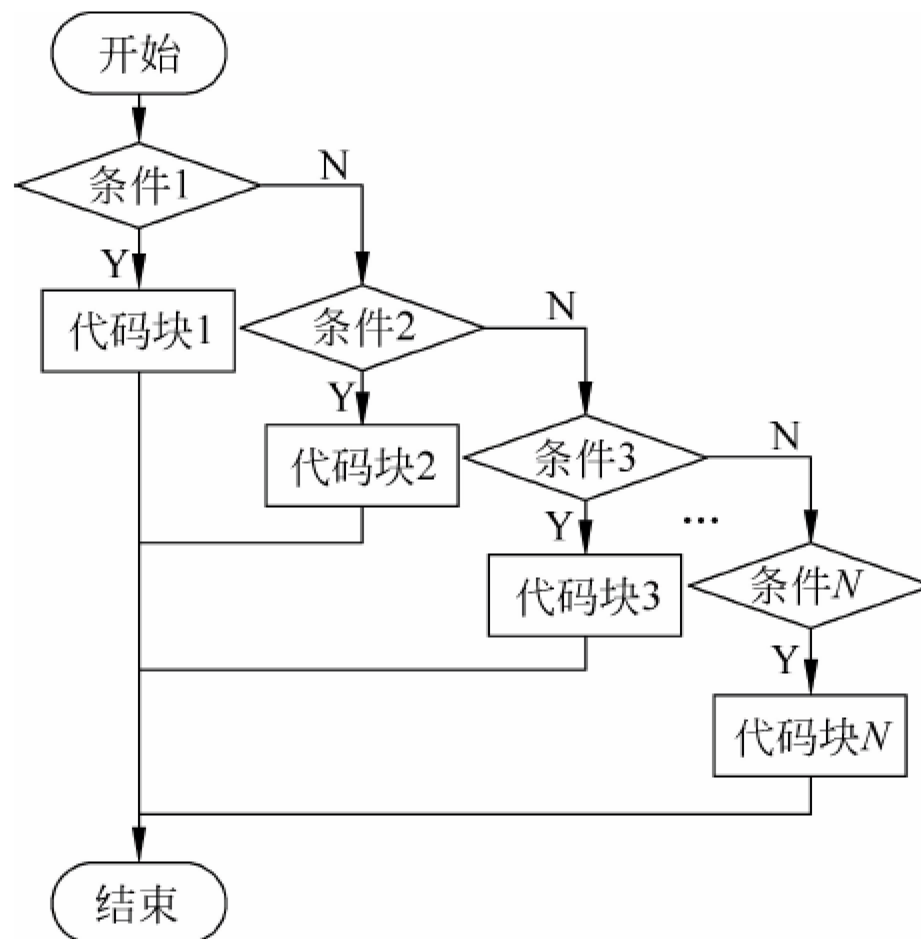


图 4-2-3　多分支执行的流程图

【例 4-2-5】　多分支求三个数的最大值。

在 Python 的 IDLE 环境下创建一个名为 if_elif_else_statement. py 的程序,代码如下所示:

```
numA = 3
numB = 4
numC = 5
if numA > = numB and numA > = numC:
    print("numA 是最大值")
elif numB > = numC and numB > = numA:
    print("numB 是最大值")
else:
    print("numC 是最大值")
```

上面就是一个非常典型的多分支条件结构,它完成了三个数求最大值的另一种思路,程序的结构往往对考虑问题的思维方式产生导向性作用,所以如果试图用一个程序解决问题的时候,在分析问题的时候,程序的结构也必须要重视。

【例 4-2-6】　多分支改进华氏摄氏温度转换问题。

已知上述几个示例中,都是单向的从一种温度转换为另一种温度,为了扩充功能,现在使用户可以自定义输入,决定是从华氏温度向摄氏温度转化,还是反之进行转化。

在 Python 的 IDLE 环境下创建一个名为 if_elif_else_temperature. py 的程序,代码如下所示:

```
print("输入 1 完成华氏转摄氏温度\n")
print("输入 2 完成摄氏温度转华氏\n")
opt = input('请输入 1 or 2: ')
if opt != "1" and opt != "2":
    print("输入错误")
elif opt == '1':
    d = input('请输入华氏温度: ')
    if float(d) <= - 459.67:
            print("输入错误")
    else:
            print("转换的摄氏温度是:")
            print(round((float(d) - 32)/1.8,2))
elif opt == '2':
```

```
d = input('请输入摄氏度: ')
if float(d) <= -273.15:
    print("输入错误")
else:
    print("转换的华氏温度是:")
    print(round((float(d) * 1.8) + 32,2))
```

程序的运行结果示例：

```
>>>
输入 1 完成华氏转摄氏温度

输入 2 完成摄氏温度转华氏

请输入 1 or 2: 1
请输入华氏温度: 444
转换的摄氏温度是:
228.89
>>>===== RESTART ========
>>>
输入 1 完成华氏转摄氏温度

输入 2 完成摄氏温度转华氏

请输入 1 or 2: 2
请输入摄氏度: 100
转换的华氏温度是:
212.0
>>>
```

以上程序的结构分别用到了双分支、多分支和嵌套。其中最外层关于 opt 的判断使用了多分支结构决定，用户到底是需要华氏转摄氏温度，还是摄氏温度转华氏温度。

4.3 用 Python 实现循环结构程序

在数学中，经常碰到类似的问题：求出 1＋2＋3＋4＋…＋100 的结果，这是一个典型的累加问题，再细看这个表达式，可以发现每项都遵循一定的数学规律，那就是逐项递增 1。在数学上可以采用梯形公式解决这个问题。计算机则可以依靠循环结构解决这种问题。如图 4-3-1 所示是为了解决这个问题设计的流程图。

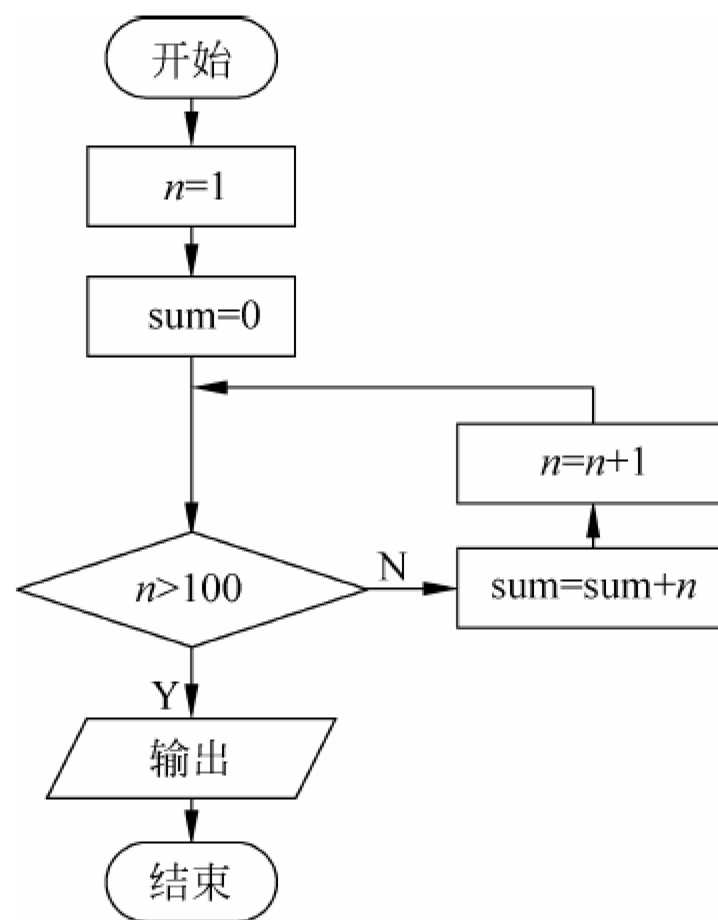


图 4-3-1　1～100 累加的流程图

在算法中，把这种从某处开始，按照一定条件，反复执行某一处理步骤的过程，叫做循环结构。它是可以循环执行某些语句的结构，能够在一定程度上减少程序的复杂性。它由循环条件和循环体组成，但存在一个上述结构都没有的安全隐患，那就是死循环。一旦循环条件设置不当，程序极有可能陷入死循环状态，这是非常危险的，它有可能耗尽系统的资源，最终导致死机。因此，为

了避免死循环的出现,在编写程序前一定要谨记下面的原则,循环体每执行一次,一定要有使得循环条件往“假”的方向发展的趋势。在本节的例子中,注意体会这一点。下面介绍几种常用的循环结构。

4.3.1 Python 的 for 循环语句

这种循环结构的语法如下:

```
for 迭代变量 in 字符串、序列或者迭代器:
    循环体
else:
    表达式
```

在循环正常退出时,会执行 else 块。

【例 4-3-1】 设计一个元组,并用循环结构依次显示其中的内容。

在 Python 的 IDLE 环境下创建一个名为 for_statement.py 的程序,代码如下所示:

```
generals = ["李元霸",
            "宇文成都",
            "秦琼",
            "程咬金",
            "单雄信" ]
print(" -- 武将列表 -- ")
for x in generals:
    print("武将姓名: ",x)
```

上述代码输出结果如下:

```
>>>
 -- 武将列表 --
武将姓名: 李元霸
武将姓名: 宇文成都
武将姓名: 秦琼
武将姓名: 程咬金
武将姓名: 单雄信
>>>
```

该实例介绍了如何对一个元组进行遍历。

【例 4-3-2】 for 循环显示字符串示例。

设计一个字符串,并用循环结构依次显示其中的内容。在 Python 的 IDLE 环境下创建一个名为 for_statement_string.py 的程序,代码如下所示:

```
mylist = "for statement"
for word in mylist:
     print(word)
else:
     print("End list")
```

程序的运行结果示例:

```
>>>
f
o
r

s
t
a
t
e
m
e
n
t
End list
>>>
```

上述结构中,每次循环迭代变量获取序列或者迭代器的当前元素,提供给循环体使用。所以循环体中,必须包含迭代变量的引用。该实例介绍了如何对一个字符串进行遍历。

【例 4-3-3】 检测字符串是否全是数字。

设计一个字符串,并用循环结构依次显示其中的内容。在 Python 的 IDLE 环境下创建一个名为 for_statement_isdigit.py 的程序,代码如下所示:

```
mystr = input("请输入一个字符串:\n")
patten_str = "0123456789"
founderr = False
for c in mystr:
    if c not in patten_str:
        founderr = True
if founderr:
    print("该字符串包含非数字字符")
else:
    print("该字符串全是数字字符")
```

程序的运行结果示例:

```
>>>
请输入一个字符串:
345454545345
该字符串全是数字字符
>>>
请输入一个字符串:
sdf4343432sdf
该字符串包含非数字字符
>>>
```

该实例在 for 结构中嵌套了一个 if 结构,用来判断循环读取的字符是否是数字。

4.3.2 Python 的 range()函数

如果需要遍历一个数字序列,可以使用 Python 中内建的函数 range()。使用它可以提供给 for 循环所需要的数字容器,例如提供一个 1～100 的数字,1～100 的偶数、奇数等。它的语法如下:

```
range(start,end,step = 1)
```

range()会返回一个包含所有 k 的列表,这里 start<=k<end,从 start 到 end,k 每次递增 step。step 不可以为零,否则将发生错误。需要注意的是 start 和 end 组成了半开区间,列表里的 k 取不到 end。

例如 range(2,19,3)

返回[2,5,8,11,14,17]

如果 range()只包含两个参数,而省略 step,step 就用默认值 1。

例如 range(3,7)

返回[3,4,5,6,]

如果 range()只包含一个参数,则该参数代表 end,start 默认为零,step 默认为 1。

例如 range(5)

返回[0,1,2,3,4]

【例 4-3-4】 遍历列表。

在 Python 的 IDLE 环境下创建一个名为 for_range_list.py 的程序,代码如下所示:

```
test_list = [1,3,4,'Hongten',3,6,23,'hello',2]
for i in range(len(test_list)):
    print(test_list[i],end = '@@')
```

运行结果:

```
>>>
1@@3@@4@@Hongten@@3@@6@@23@@hello@@2@@
>>>
```

【例 4-3-5】 在 Python 的 IDLE 环境下创建一个名为 for_range.py 的程序,代码如下所示:求 1～100 的所有偶数的和。

思维目标:

迭代思维

判别一个数是否是偶数的思路

```
sum = 0
for x in range(0,101,1):
   if x % 2 == 0:
        sum = sum + x
print("累加和是:",sum)
```

执行结果示例:

```
>>>
累加和是: 2550
>>>
```

该程序在执行后，将会输出 1～100 的所有偶数的和。读者要掌握程序中通过 $x \% 2==1$ 语句判断一个数是不是偶数的方法。

思维扩展：

(1) 请读者考虑，为了完成同样的功能，如何对上述程序进行修改，使得程序效率更加高。

(2) 请读者利用相同的迭代思维，求出 10 的阶乘。

【例 4-3-6】 对鸡兔同笼问题进行求解，如图 4-3-2 所示。



图 4-3-2　参考图

在 Python 的 IDLE 环境下创建一个名为 chickenAndRabit.py 的程序，代码如下所示：

思维目标：穷举法思维

```
numHeads = 35
numLegs = 94
for numChickens in range(0, numHeads + 1):
    numRabits = numHeads - numChickens
    if 2 * numChickens + 4 * numRabits == numLegs:
        print("numChickens:",numChickens)
        print("numRabits:",numRabits)
```

程序的运行结果：

```
>>>
numChickens: 23
numRabits: 12
>>>
```

思维扩展：

(1) 请考虑，如何将该方法抽象为解决所有二元一次方程组的思路。

(2) 如何从通用化的角度考虑，解决同样的问题可以尽可能减少迭代的次数。

(3) 穷举思路。

4.3.3 Python 的 while 循环结构

在许多情况下,当一个循环执行之前,可能并不知道它需要执行的次数。这时,就可以使用 while 循环。这种循环结构的语法是

```
while 循环条件:
    循环体
```

在上述结构中,尤其要注意死循环的问题。因为这种循环与上面的 for 结构不同,for 结构由一个循环变量来控制循环的次数,而 while 循环结构则只能依靠循环条件是否成立来判断是否退出循环。所以,在 while 循环结构中,如果要避免死循环,就必须在循环体中加入合理的语句。通过这个语句,在每次循环得到执行后,都能使得循环条件往不成立的方向前进。如图 4-3-3 所示是该循环结构的流程图。

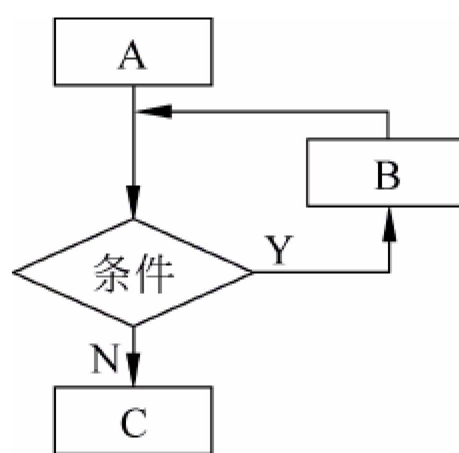


图 4-3-3　while 循环流程图

从该流程图可以看出:

(1) 只有一个入口。

(2) 只有一个出口。(请注意:一个菱形判断框有两个出口,而一个选择结构只有一个出口。不要将菱形框的出口和选择结构的出口混淆。)

(3) 结构内的每一部分都有机会被执行到。

(4) 结构内不存在“死循环”(无终止的循环)。

【例 4-3-7】 求解银行存款利息问题。

已知银行存款利率为 1.9%,编写程序计算并输出需要存多少年,10 000 元的存款本金才会连本带利翻一番。这显然是一个迭代问题,而且是以年为单位的,本金在发生变化,而这样的循环究竟要循环多少次呢?显然是这个问题要求解的结果,对于不知道循环次数的问题,使用 while 是非常方便的。

在 Python 的 IDLE 环境下创建一个名为 while_statement.py 的程序,代码如下所示:

```
cunkuan = 10 000   # 本金 10 000 元
years = 0
while cunkuan < 20000:
    years += 1
    cunkuan = cunkuan * (1 + 0.019)
print(str(years) + "年以后,存款会翻番")
```

程序运行结果如下:

```
>>>
37 年以后,存款会翻番
>>>
```

4.3.4 Python 的 break、continue 和 pass 语句

在循环过程中,可使用循环控制语句来控制循环的执行。有三个控制语句,分别是

break、continue 和 pass。一般说来，break 和 continue 语句的作用是改变控制流程。当 break 语句在循环结构中执行时，它会忽视后面的代码块，立即跳出其所在的最内层的循环结构，转而执行该内层循环结构后面的语句。与 break 语句不同，当 continue 语句在循环结构中执行时，并不会退出循环结构，而是立即结束本次循环，重新开始下一轮循环，也就是说，跳过循环体中在 continue 语句之后的所有语句，继续下一轮循环。对于 while 语句，执行 continue 语句后会立即检测循环条件；对于 for 语句，执行 continue 语句后并没有立即检测循环条件，而是先将“可遍历的表达式”中的下一个元素赋给控制变量，然后再检测循环条件。下面通过几段简单的代码，了解下它们的工作过程。

```
mylist = ["zope","Python","perl","Linux"]
for technic in mylist:
  if technic == "perl":
            break
  else:
           print technic
```

上述代码的运行结果如下：

```
zope
Python
```

continue 语句会忽略后面的语句，强制进入下一次循环。例如：

```
mylist = ["zope","Python","perl","Linux"]
      for technic in mylist:
        if technic == "perl":
            continue
        else:
            print technic
```

上述代码的运行结果如下：

```
zope
Python
Linux
```

讨论：请讨论上述两段代码，体会退出循环和退出该次循环的区别。

循环体可以包含一个语句，也可以包含多个语句，但是却不可以没有任何语句。那么，如果只是想让程序循环一定次数，但是循环过程什么也不做的话，那该怎么办呢？使用 pass 语句，它不执行任何操作。示例：

```
for technic in mylist:
    if technic == "perl":
            pass
    else:
            print technic
```

上述代码的执行结果示例如下：

```
zope
Python
Linux
```

再如:

```
While1:
pass#
```

这段代码就可以实现等待键盘中断。

【例 4-3-8】 在已知的一个序列中,查找某个数字,当找到的话输出“找到了”,并停止查找。在 Python 的 IDLE 环境下创建一个名为 for_break_statement.py 的程序,代码如下所示:

```
numbers = [100, 25, 125, 50, 150, 75, 175]
for x in numbers:
  print(x)
  if x == 50:
      print("找到了")
      break;
```

程序的运行结果示例:

```
>>>
100
25
125
50
Found It!
>>>
```

在该实例中,break 语句可以省略,并不影响程序结果,但是却大大影响了程序的效率,如果有 break 语句程序,在找到 50 后就会停止查找,反之,程序会继续往下查找浪费资源。

4.3.5 循环结构应用

1. 三角星号的输出

将一个星号(*)按照一定的规律输出,组成三角形、菱形等图形,是学习循环结构经常用到的应用案例。

【例 4-3-9】 输出如图 4-3-4(a)所示的星号。

基本思路:通过观察上图,不难发现总共输出 10 行,每行输出若干星号,而且星号的数量和行号之间有一个函数关系,那就是星号的数量等于行号。这对于程序设计结构,可以通过双重循环实现,一重循环输出行,一重循环输出某一行的星号。假设 i 代表行号,则 i 的取值范围是 1～10,j 代表每行星号的数量,对于第 i 行而言,j 的取值范围为 1～i。根据上述分析,在 Python 的 IDLE 环境下创建一个名为 star1.py 的程序,代码如下所示:

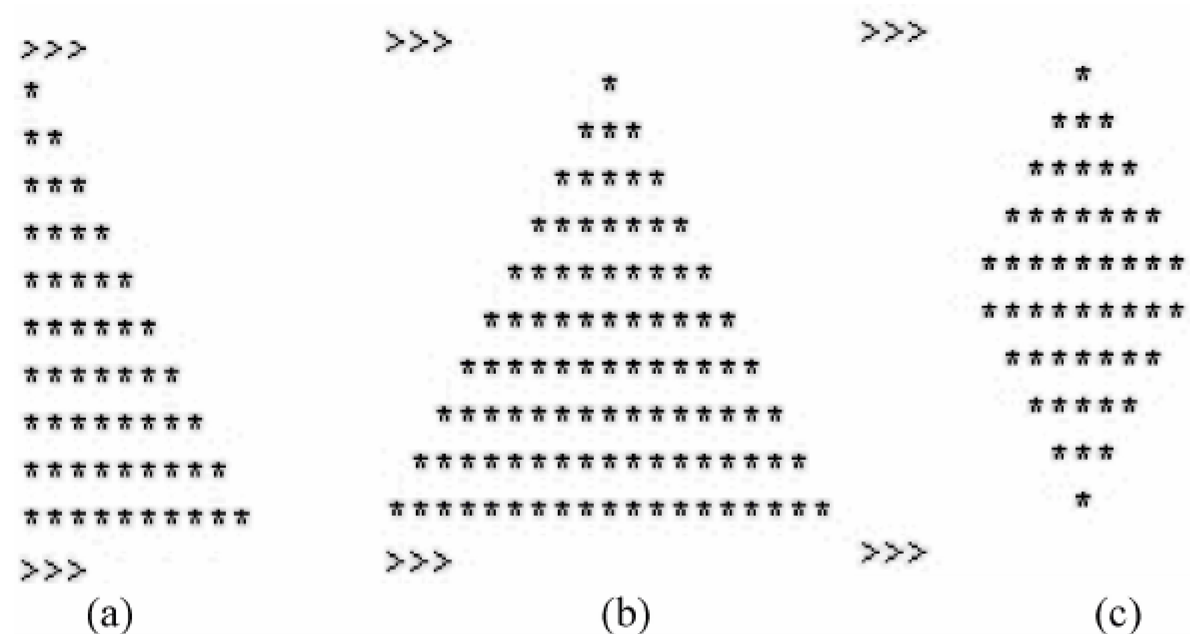


图 4-3-4　输出星号示意图

```
for i in range(1,11):
    s = ""
    for j in range(0,i):
        s += "*"
    print(s)
```

在掌握了字符串的＊操作后,该实例还可以用更简单的方式实现。在 Python 的 IDLE 环境下创建一个名为 star2.py 的程序,代码如下所示:

```
for i in range(1,11):
    print("*" * i)
```

【例 4-3-10】 输出如图 4-3-4(b)所示的星号。

根据例 4-3-9 的分析,该题仍然采用双重循环,所不同的是,具体到每一行,i 和 j 的函数关系发生了变化,该题中,对于第 i 行,前面要先输出 $10-i$ 个空格,再输出 $2\times i-1$ 个星号。在 Python 的 IDLE 环境下创建一个名为 star3.py 的程序,代码如下所示:

```
for i in range(1,11):
    s = ""
    for  j in range(0,10 - i):
        s += " "
    for  j in range(0,2 * i - 1):
        s += "*"
    print(s)
```

思维扩展:

按照前面两个例子的分析,编写程序输出图 4-3-4(c)的图形。

2. 约瑟夫环问题

据说著名犹太历史学家约瑟夫有过以下的故事:在罗马人占领乔塔帕特后,39 个犹太人与约瑟夫及他的朋友躲到一个洞中,39 个犹太人决定宁愿死也不要被敌人抓到,于是决定了一个自杀方式,41 个人排成一个圆圈,由第一个人开始报数,每报数到第三人该人就必须自杀,然后再由下一个重新报数,直到所有人都自杀身亡为止。然而约瑟夫和他的朋友并不想遵从,约瑟夫要他的朋友先假装遵从,他将朋友与自己安排在第 16 个与第 31 个位置,于是逃过了这场死亡游戏。

后来大家把这个问题归结为一个数学的应用问题:已知 n 个人(以编号 $1,2,3,\cdots,n$ 分

别表示)围坐在一张圆桌周围。从编号为 k 的人开始报数,数到 m 的那个人出列;他的下一个人又从 1 开始报数,数到 m 的那个人又出列;依此规律重复下去,直到圆桌周围的人全部出列,这个问题也被称为约瑟夫环问题。

【例 4-3-11】 一共有 30 个人,从 1~30 依次编号。每个人开始报数,报到 9 的人自动出列,当有人出列后,从后一个人开始重新从 1 报数,以此类推。求出列的前 15 个人的号码。

思路分析:该题目选择合适的数据结构很重要,为了使得出列的人不再重复报数,专门使用一个 list 数据结构存储出列的人,每个人报数前,先到该 list 中查找是否已出列,如果出列就不报数,否则报数。

在 Python 的 IDLE 环境下创建一个名为 joseph.py 的程序,代码如下所示:

```
mylist = range(1,31)                    #总共 30 个数
delete_list = []                        #出列的人
x = 0                                   #人的下标
i = 0                                   #代表报数
while len(delete_list)<15:
  if(x > 29):
        x = 0
  if(mylist[x] not in delete_list):     #人不在出列中
      i = i + 1                         #依次报数
  if i == 9:
    print(mylist[x])
    print("出列: ")
    delete_list.append(mylist[x])
    i = 0
  x = x + 1
```

程序运行结果:

```
>>>
9
出列:
18
出列:
27
出列:
6
出列:
16
出列:
26
出列:
7
出列:
19
出列:
30
出列:
```

```
12
出列:
24
出列:
8
出列:
22
出列:
5
出列:
23
出列:
>>>
```

思维扩展：该实例可以让读者清楚求模运算的作用。请利用该例子，对于下面的问题，设计程序进行求解。

一个监狱要枪毙100个犯人。但是有一个奇怪的规定，所有犯人排成一排，编号1、2、…，第一次枪毙单数，剩下的继续编号，再枪毙单数。问最后剩下的是几号？

4.4 字符串数据操作

在第3章中，已经介绍了Python的数据类型，其中字符串数据类型，又称为文本类型，它在Python中使用频率很高，本节利用前面讲述的控制结构，专门介绍下针对这种数据类型的操作。

4.4.1 字符串和List数据的相互转换

在编程的时候，经常会出现字符串和List的转换，熟练掌握它们之间的互相转换，往往可以轻松地解决一些看似很棘手的问题。

【例4-4-1】 某字符串mystr存储了一个学生的所有兴趣爱好，代码如下：

```
mystr = "篮球、足球、乒乓球、羽毛球、电子游戏、看书、旅游、电影、音乐"
```

如何能快速地知道该生有多少项爱好？已知该生的兴趣爱好之间是用顿号进行分割的。

在Python的IDLE环境下创建一个名为stringTolist.py的程序，代码如下：

```
mystr = "篮球、足球、乒乓球、羽毛球、电子游戏、看书、旅游、电影、音乐"
li = mystr.split('、')
print("该生的爱好有" + str(len(li)) + "项")
```

程序运行结果如下：

```
>>>
该生的爱好有9项
>>>
```

字符串通过调用 split()函数,可以将自身转换为 list 数据类型,这样就可以通过 len 函数获取 list 的项数,从而得到本题的答案。

【例 4-4-2】 某字符串 mystr 存储的内容是一串数字 0122202341020303,现在要求把该数偶数位置上的数字用“-”代替。

在 Python 的 IDLE 环境下创建一个名为 stringTolist2.py 的程序,代码如下:

```
mystr = "0122202341020303"
print(mystr + "\n")
li = list(mystr)
for i in range(1,len(li),2):
     li[i] = '-'
mystr = "".join(li)
print(mystr)
```

程序运行结果如下:

```
>>>
0122202341020303

0-2-2-2-4-0-0-0-
>>>
```

字符串虽然支持遍历操作,但是字符串的元素不能被修改,所以可以借助 List 数据类型,通过语句 li=list(mystr),把字符串 mystr 强转为 li 后,利用 List 可以修改元素内容的特点,再把 li 通过"".join(li)语句还原回原来的字符串。

4.4.2 字符查找

如果需要在某个字符串中查找特定的字符,既可以通过对字符串进行遍历,又可以通过字符串的函数处理。

【例 4-4-3】 在一个输入的字符串中查找是否有字符“梦”存在。在 Python 的 IDLE 环境下创建一个名为 FindChar.py 的程序,代码如下:

```
while True:
    str = input("请输入一个字符串(quit 退出)\n")
    if(str == "quit"):
        break
    else:
        findok = False
        for mychar in str:
          if mychar == "梦":
              findok = True
              break
        if findok:
            print("找到字符梦")
        else:
            print("没找到字符梦")
```

程序运行结果如下:

```
请输入一个字符串(quit 退出)
我的中国梦
找到字符梦
请输入一个字符串(quit 退出)
我的中国心
没找到字符梦
请输入一个字符串(quit 退出)
```

该实例是典型的字符串遍历思路，通过循环结构将字符串中包含的所有字符一一取出，和字符“梦”作匹配。利用字符串函数也可以将上述实例改为以下的代码：

```
while True:
    str = input("请输入一个字符串(quit 退出)\n")
    if(str == "quit"):
        break
    else:
        if str.count('梦')!= 0:
            print("找到字符梦")
        else:
            print("没找到字符梦")
```

因为字符串的函数 count()是统计某个字符在字符串中出现的次数，所以上述代码利用了这个原理，通过语句 if str.count('梦')!=0 判断字符串中是否包含字符“梦”。请读者自行考虑，如果使用字符串的函数 find()能否实现特定字符的查找。

4.4.3 字符串遍历

例 4-4-3 介绍了字符串遍历，但是 for 字符 in 字符串这种类型的遍历是一种无序的遍历，即这种遍历只强调了字符有没有包含在字符串中，但是字符在字符串的哪个位置是无所谓的。接下来就介绍字符串的有序遍历。

【例 4-4-4】 输入任何一个数字，如果数字中的每位数字之和(最高位除外)，等于最高位上的数，则输出找到了，否则输出找不到。在 Python 的 IDLE 环境下创建一个名为 string_traversal.py 的程序，代码如下：

```
while True:
    sum = 0
    str = input("请输入一个数字(quit 退出)\n")
    if(str == "quit"):
        break
    else:
        for i in range(1,len(str)):
          sum = sum + int(str[i])
        if sum == int(str[0]):
            print("找到了")
        else:
            print("找不到")程序运行结果:
```

程序的输出结果如下：

```
>>>
请输入一个数字(quit 退出)
8233
找到了
请输入一个数字(quit 退出)
712
找不到
```

该实例是典型的字符串有序遍历,通过 for 和 range 实现了对字符串的逐一遍历。

4.4.4 字符串截取

在实际生活中,一个产品的编码、一个居民身份证、一个学号都可以使用字符串数据类型表示,然而这几种数据,往往蕴含着很多编码信息,例如,产品编码的某几位可能蕴含产品类别信息,居民身份证更是蕴含出生年月、出生地区、性别等信息。如果从已知的字符串中,把它所蕴含的信息识别出来,这往往需要用到字符串截取。

【例 4-4-5】 已知某个产品的编码是 2320060214-345,该编码信息如下:

第 1 位为 1 表示该商品在市,为 2 表示该商品退市;

第 2 位表示商品的类别;

第 3～第 10 位表示商品的出厂日期;

第 12～第 14 位表示商品的编号;

现在输入一个符合上述规定的产品编码,通过程序将其是否在市以及出厂日期识别出来。

在 Python 的 IDLE 环境下创建一个名为 string_traversal. py 的程序,代码如下:

```python
while True:
    mystr = input("请输入一个产品编码(quit 退出)\n")
    if(mystr == "quit"):
        break
    else:
        if mystr[0] == "1":
            print("商品在市,")
        else:
            print("商品退市,")
        myyear = mystr[2:6] + "年"
        mymonth = mystr[6:8] + "月"
        myday = mystr[8:10] + "日"
        print("商品的出厂日期是" + myyear + mymonth + myday)
```

程序的运行结果如下:

```
>>>
请输入一个产品编码(quit 退出)
2320060214 - 345
商品退市,
商品的出厂日期是 2006 年 02 月 14 日
请输入一个产品编码(quit 退出)
```

该实例是典型的字符串截取，通过操作符[]+索引区间实现字符串的截取，需要注意的是，[]是一个左闭右开区间，例如mystr[2:6]意思是从第2位取到第5位，取不到第6位，还有就是Python支持索引为负数，例如mystr[-1]代表该字符串的最后一位。

4.5 本章小结

本章重点介绍了程序设计中所涉及的基本结构，包括它们的概念、流程图以及相应的实例。还对字符串数据类型操作进行了归纳和分类，并分别通过实例对其进行了介绍。

本章要点如下：

(1) 顺序结构的流程图以及相应的实例。

(2) if:结构的语法重点和实例。

(3) if:…else:结构的语法重点和实例。

(4) if:…elif:…else:结构的语法重点和实例。

(5) for…in…结构的语法重点和实例，此结构可以遍历序列、元组和字符串。

(6) range()的语法重点和实例。

(7) for和range()结合进行计数循环。

(8) while:结构的语法重点和实例，以及while和for在使用上的区别。

(9) break、continue、pass三种控制结构在提升程序运行效率上的作用。

(10) 循环结构的典型应用：三角星号的输出、约瑟夫环问题。

(11) 字符串和list数据类型的相互转换，以及split()函数和join()函数的使用方法。

(12) 字符串查找中涉及的count()和find()函数的使用方法。

(13) 字符串的截取，掌握Python中通过索引对字符串截取的方法。

4.6 习题与思考

4-1 continue短语用于________。

A. 退出循环程序　　　　B. 结束本次循环

C. 空操作　　　　D. 根据if语句的判断进行选择

4-2 在Python语句：for i in range(10):……中，循环终值是________。

A. 9

B. 10

C. 11

D. 语句错误，range()函数需有初值和终值两个数据

4-3 在以下Python循环中：

for i in range(1,3):(换行并缩进)for j in range(2,5):(换行并缩进)print(i*j)。语句print(i*j)共执行了________次。

A. 2　　B. 3　　C. 5　　D. 6

4-4 已知有二元一次方程组：

x + y = 12

```
2x - 3y = 14
```

请利用求解鸡兔同笼问题的思路,将其中的 x,y 的值求出。

4-5　请以自己的身份证为例,识别出它所包含的出生日期信息和性别信息,输出格式如下:

您的出生日期为××××年×月×日,性别为×

4-6　请读者思考,如何修改例 4-3-6 中的鸡兔同笼问题,使得它的执行效率更高。

4-7　写一个程序,用户输入行数,能根据行号输出星号,结果如图 4-6-1 所示(不允许使用双重循环)。

```
  *
 ***
*****
```

图 4-6-1　题 4-7 图

4-8　随机输入 10 个不重复的正整数,然后输出排序后的数列。

具体要求:

每次输入一个数前,都提示"请输入第 X 个数字"。

如果输入的数字有重复,提示"数字有重复"之后,继续提示"请输入第 X 个数字"。

如果正确输入完第 10 个数字之后,显示从小到大排序后的 10 个数字的列表。

【提示:可使用 while 结构和 list 列表,X 代表具体的第几个数。】

4.7　实验　基本控制结构

4.7.1　实验目标

(1) 掌握基本程序设计的结构。

(2) 掌握分支结构,并能利用它解决实际问题。

(3) 掌握 for 循环结构,并能利用它解决实际问题。

(4) 掌握 while 循环结构,并能利用它解决实际问题。

(5) 掌握 range 函数的使用方法,并能结合 for 一起使用。

(6) 掌握 break、continue 和 pass 语句的使用,能够优化一些实际问题的解决方案。

4.7.2　实验范例

1. 分支结构范例

从屏幕输入一个数字,和系统已经设置的数比较大小,如果和预设的数字相等,则显示相等,如果比预设的数太小,则显示数字太小,否则显示数字太大。

(1) 首先确定该问题涉及的结构,是双分支结构。

(2) 然后确定输入和输出。

(3) 核心内容是比较,将系统的数字和输入的数字分别放入两个变量,然后利用比较运算符==比较。

(4) 编码如下:

```
number = 23
guess = int(input('请输入一个数 : '))
if guess == number:
```

```
    print '相等.'
elif guess < number:
    print '数字太小
else:
    print '数字太大'
```

2. 循环结构范例

假设有个文本文件 A.txt 内容如图 4-7-1(a)所示，使用列表，计算文件中 5 组数据（每组两个）的和、差、积和商，计算结果写入另一个文件 B.txt，结果如图 4-7-1(b)所示。

A.txt

```
45    634.6
783   73
233   45
6     34.6
78    84
```

(a)

B.txt

```
45+634.6=679.6
45-634.6=-589.6
45*634.6=28557.0
45/634.6=0.0709108099590293
783+73=856.0
783-73=710.0
783*73=57159.0
783/73=10.726027397260275
233+45=278.0
233-45=188.0
233*45=10485.0
233/45=5.177777777777778
```

(b)

图 4-7-1　文本文件内容

(1) 首先确定该程序的主题结构应该是循环。

(2) 该程序的输入和输出，不再是基于屏幕的，而是使用文件的读写功能，从 A.txt 中读取数据，将结果输入 B.txt 文件。

(3) 为了体现出循环的优势，可以先用顺序结构编写，然后再改为循环，对比循环的优势。

(4) 编码示例如下：

```
f = open('c:\\Python32\\b.txt','w')
numbers = list(open('c:\\Python32\\a.txt','r'))  # 将 A.txt 文件中的所有行读入 numbers
                                                 # 这个列表中
x = numbers[0].split()                           # 将第一行的数据分解
y1 = float(x[0]) + float(x[1])
y2 = float(x[0]) - float(x[1])
y3 = float(x[0]) * float(x[1])
y4 = float(x[0])/float(x[1])
s = x[0] + '+' + x[1] + '=' + repr(y1) + '\n'
s = s + x[0] + '-' + x[1] + '=' + repr(y2) + '\n'
s = s + x[0] + '*' + x[1] + '=' + repr(y3) + '\n'
s = s + x[0] + '/' + x[1] + '=' + repr(y4) + '\n'
f.write(s)
x = numbers[1].split()                           # 将第二行的数据分解
```

```
y1 = float(x[0]) + float(x[1])
y2 = float(x[0]) - float(x[1])
y3 = float(x[0]) * float(x[1])
y4 = float(x[0])/float(x[1])
s = x[0] + ' + ' + x[1] + ' = ' + repr(y1) + '\n'
s = s + x[0] + ' - ' + x[1] + ' = ' + repr(y2) + '\n'
s = s + x[0] + ' * ' + x[1] + ' = ' + repr(y3) + '\n'
s = s + x[0] + '/' + x[1] + ' = ' + repr(y4) + '\n'
f.write(s)
… #将第三行分解
… #将第四行分解
f.close()
```

如果使用该章节的 for 循环，程序可以改造为

```
f = open('c:\\Python32\\b.txt','w')
numbers = list(open('c:\\Python32\\a.txt','r'))
n = len(numbers)
for i in range(n):
    x = numbers[i].split()
    y1 = float(x[0]) + float(x[1])
    y2 = float(x[0]) - float(x[1])
    y3 = float(x[0]) * float(x[1])
    y4 = float(x[0])/float(x[1])
    s = x[0] + ' + ' + x[1] + ' = ' + repr(y1) + '\n'
    s = s + x[0] + ' - ' + x[1] + ' = ' + repr(y2) + '\n'
    s = s + x[0] + ' * ' + x[1] + ' = ' + repr(y3) + '\n'
    s = s + x[0] + '/' + x[1] + ' = ' + repr(y4) + '\n'
f.write(s)
f.close()
```

程序利用了循环结构之后，可以从内容上看到，思路更加简洁，而且程序不易因为赘述大量雷同的内容，而增加出错的概率。

3. range()函数的使用举例

1）列举某个数内的所有数

```
for i in range(5):
  print(i,end = ',')
```

运行结果如下：

```
>>>
0,1,2,3,4,
>>>
```

上述代码就使用到了 Python 中的内置函数 range(5)，其中参数 5 代表：0～4 的一个序列。

2）自定义起点和终点的序列

```
print('range(5,100)表示：',range(5,100))
listB=[i for i in range(5,100)]
print(listB)
print('####################################################')
```

运行结果如下：

```
>>>
range(5,100)表示：range(5, 100)
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
####################################################
>>>
```

上述代码定义了一个从 5 开始的起始点，到 100 结束的结束点，注意区间是左闭右开。

3）带步长的序列

除此之外，还可以定义步长。例如定义一个从 1 开始到 30 结束，步长为 3 的列表。

```
print('range(1,30,3)表示：',range(1,30,3))
listC=[i for i in range(1,30,3)]
print(listC)
```

运行效果如下：

```
>>>
range(1,30,3)表示：range(1, 30, 3)
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28]
>>>
```

（1）持续地让用户从屏幕输入一个数字，直到该数字和预设数字相等为止，此时输出相等，如果和系统已经设置的数相比太小，则显示数字太小，并提示用户继续输入，否则显示数字太大，并提示用户继续输入。

① 题目分析：为了实现持续地提示用户输入数字，把语句 input 和上述实验范例 1 中的 if 语句放到 while true 循环中。这样除非程序会遇到 break 语句，否则将一直循环输入。请读者掌握这种开关变量的用法。

② 参考代码如下：

```
number = 23
while True:
    guess = int(input('请输入一个数：'))
    if guess == number:
```

```
            print('相等')
            break
        elif guess < number:
            print('数字太小')
        else:
            print('数字太大')
print ('循环结束')
```

(2) 求 10 之内素数,如果该数是素数请输出 X 是素数,如果该数是合数,请输出它的因子相乘的式子,例如 4=2×2。

① 首先了解素数以及合数的定义。

② 确定程序的整体架构,因为知道边界条件可以确定使用 for 循环。

③ 参考代码如下:

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
          print(n, 'equals', x, '*', n//x)
          break
    else:
        print(n, 'is a prime number')
```

④ 程序运行结果如下:

```
>>>
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
>>>
```

4.7.3 实验内容

(1) 输入某年某月某日,判断这一天是这一年的第几天。

程序分析:以 3 月 5 日为例,应该先把前两个月的加起来,然后再加上 5 天即本年的第几天,特殊情况,闰年且输入月份大于 3 时需考虑多加一天。

(2) 在 Python IDLE 中输入 list(range(5,21,4)),查看结果是什么。

(3) 在 Python IDLE 中输入 list(range(3,7)),查看结果是什么。

(4) 请输入任意一个字符串,通过程序判断出该字符串中的字符是否全为英文字母。

(5) 请利用多分支结构,编写 4 个数求最大值的程序。

(6) 持续让用户输入，每输入一个字符串，程序输出字符串的长度，直到输入 quit 字符串，程序终止运行。

(7) 随机输入一个字符串，把最右面的 10 个不重复的字母（不区分大小写）挑选出来。具体要求如下：

如果输入的字符串中找不出 10 个字母，要显示提示信息“找不到 10 个字母”。

如果找到后，请输出包含它们的序列。

第5章 数据的输入和输出

程序设计中最常用的输入输出方式为标准输入输出和文件输入输出。标准输入输出是使用键盘输入数据，结果显示在屏幕上的方式。文件输入输出则是通过程序访问文件来完成数据读取和写入的方式。

异常是程序运行时发生的错误，如用户的非法输入、除数为零、打开的文件不存在等。这类错误很容易发生在程序交互处理的过程中，所以将异常处理放在本章中加以介绍，重点说明异常是如何发生的、异常的名称、“默认异常处理器”的作用以及在程序中如何控制异常。

5.1 人-机交互的意义、方法

“编程作为一种智力活动，它是唯一一种能让你创造出交互式艺术作品的艺术形式。你创造出来人们可以操作的软件，你是在间接地和人们交互。没有任何其他艺术形式有如此的交互性。电影是单向地向观众传输信息，绘画是静态的，而软件程序却是双向动态的。”——摘自 *Learn Python The Hard Way, 2nd Edition* 这本书的尾声部分。

程序的交互性可以体现为两种形式：键盘交互和文件交互。键盘交互是指通过键盘输入数据，在显示屏上显示结果，也称为“标准输入输出”。文件交互是指程序从数据文件中读取数据，运行后还可以将结果写入一个文件中的交互方式。

5.1.1 标准输入输出

计算机的标准输入输出设备是键盘和鼠标，标准输入输出就是指通过键盘输入数据，在显示屏上显示结果的过程。

1. 标准输入操作

标准输入操作是指通过键盘的键入，输入数据到程序。

对键盘的每一个按下动作，都会通过键盘的电路转化为一个字符送到键盘缓冲区，所以无论按下的是 a，还是 1，都是字符而不会是其他数据类型，程序在接受数据的时候，需要按照程序的需要，将文本数据转化为相应的数据类型。程序语言使用不同的方法将读入的文本数据转化为相应的数据类型，有的是通过格式化输入函数，有的是通过类型自动转化。

2. 标准输出操作

标准输出操作是指将程序的数据显示到计算机的标准输出设备——显示屏上。显示屏的输出是按输出字符调用它的字形码，再按字形阵列显示图形，所以无论程序的数据是什么数据类型的，都要先转化成字符。

同样，程序语言使用不同的方法将程序数据转化为字符，有的是通过格式化输出函数，有的是通过类型自动转化，还有的是通过文本格式化函数将数据转化为字符串。

3. 人机交互的方式

使用标准输入输出构造人机交互最基本的算法过程为以下三步：

(1) 输入数据。

(2) 处理数据。

(3) 输出数据。

【例 5-1-1】 重新考虑第 3 章中例 3-1-6 求三角形面积的程序，将其中的对三角形三边赋值的语句修改为输入语句，在程序运行时，由用户按需输入一个三角形的三边，就可以实现求任意一个三角形的面积了，算法设计如下：

```
1. 显示："输入第一条边"。
2. 接收键盘输入的数到变量 a 中。
3. 显示："输入第二条边"。
4. 接收键盘输入的数到变量 b 中。
5. 显示："输入第三条边"。
6. 接收键盘输入的数到变量 c 中。
7. 计算 s = (a + b + c)/2。
8. 计算面积 area。
9. 显示面积。
```

其中第 1～第 6 步是输入数据部分，第 7～第 8 步是计算数据的部分，第(9)步是输出数据部分。

在实际遇到的程序实践中，这三部分不一定是分离的，有时输入和处理融合在一起，边输入边处理，有时处理和输出融合在一起，边处理边输出。

【例 5-1-2】 输入和处理融合在一起示例，计算输入的一组整数之和，算法设计如下：

```
1. 显示：n = 。
2. 接收键盘输入的数到变量 n 中。
3. 累加器置零 sum = 0。
4. 循环迭代 i 从 1 到 n。
   4.1  接收键盘输入的数到变量 x 中。
   4.2  将 x 累加到 sum。
5. 显示累加和 sum。
```

在此例中，一组整数的输入和累加计算由一个循环操作控制，每输入一个数，就计算一次。

5.1.2 文件输入输出

1. 文件和文件目录

在程序设计中，文件是一些具有永久存储特性及特定顺序的字节组成的一个有序的，具有名称的集合，它保存在磁盘、光盘、磁带等存储设备上。标准输入输出的数据是内存存储的数据，内存数据在程序执行结束后就不复存在了。对于大批量的原始数据和处理后的结果数据通常需要长久保存，可以保存在文件中。

通常情况下，文件按树状目录结构进行组织，为了管理一个文件，一般需要制定驱动器、目录路径和文件名。例如：C:\sample\ch5\5-1-3.py 表示存放在 C 盘中文件夹 sample\ch5 下的文件 5-1-3.py，这种表示方法称为绝对路径文件名。

文件还可以以相对路径文件名的方式表示，相对于当前路径，从当前路径出发表示一个文件。例如，当前路径为 C:\sample，文件 5-1-3.py 可以表示为 ch5\5-1-3.py，如果当前路径为 C:\sample\ch5，直接读取文件名 5-1-3.py 即可。

一个执行的程序的当前路径就是该程序所在的文件夹，如果处理的数据文件与 Python 程序在同一文件夹下，可以直接用文件名表示。如果处理的数据文件在不同的文件夹下，通常要用绝对路径文件名表示。

程序语言一般会按对象和流的方式来管理文件及文件目录。

2. 访问文件的流程

访问文件的流程一般为：

(1) 打开文件。

(2) 访问文件(读/写)。

(3) 关闭文件。

程序可访问的数据文件分为二进制文件和文本文件，二进制文件直接按数据的二进制编码组织数据，文本文件逐个字符存储字符的 ASCII 编码。打开文件可以做的操作有读、写、追加等，打开文件时要明确文件的访问方式，打开后按访问方式有限制地访问文件，若要改变文件的访问方式，必须先关闭文件，再次按新的访问方式打开文件。

访问文件包括将文件中的数据读取存储到内存中去和将存储在内存中的程序数据写入文件中。通常文本文件是由字符构成的，没有数据类型，在读入时与标准输入相似，由处理文件的程序确定数据类型。

文件是流式结构或顺序结构，所以在顺序地读取文件中的字符串时，程序还要将读入的字符串转化为程序所需的数据类型，甚至构造所需的复杂的数据结构(链表、队列、树等)。写文件时，组织程序数据顺序地写入文件中。

文件交互的优势在于可以处理大批量的数据，并在文件中长久地保存计算结果。

5.2　标准输入输出程序

5.2.1　标准输入函数

Python 提供内置的 input()函数，用于在程序运行时接收用户的键盘输入。input()语句的基本格式为

```
input('提示文本串')
```

执行 input 语句时先输出提示文本串，在输入的时候起辅助作用，提示用户需要输入怎样的数据，可以省略。当用户输入一个数据并按 Enter 键后，input 函数返回数据的字符串对象，也就是说无论输入的数据是整数，还是浮点数，或字符串，从 input 函数得到的都是字符串。

【例 5-2-1】 输入一个整数数据示例，通常需要用一个变量来接收用户输入的数据。

```
>>> data = input('请输入一个整数: ')
请输入一个整数: 34
>>> data
'34'
```

带下划线的部分表示用户输入的数据。程序执行到该语句,会等待用户通过键盘输入数据,当用户输入数据 34 并按 Enter 键后,input 函数将用户输入的数据作为一个字符串返回,并赋给变量 data。所以在这里,data 是一个字符串变量,如果要使 data 变量作为整型变量参加以后的算术运算,可采用以下语句、在接收输入的同时进行类型转换:

```
>>> data = int(input('请输入一个整数: '))
请输入一个整数: 34
>>> data
34
```

程序运行过程中程序显示一个输入提示,用户输入一个数据, int 函数将用户输入的字符串转化为 int 实数类型。

在有些用户交互场合下存在着一些特殊的表示习惯,如输入时间,习惯表示为 7:30:25,表示 7 点 30 分 25 秒,在程序中需要使用 hour、minute、second 三个变量接收三个整数值,输入的字符串需要进行分割处理。

【例 5-2-2】 时间的输入示例。

```
>>> hour,minute,second = input('请输入一个时间(h:m:s) :').split(':')
请输入一个时间(h:m:s):7:30:25
>>> hour,minute,second
('7', '30', '25')
```

input 函数的返回值是一个字符串'7:30:25',对字符串对象调用 split 方法按冒号':'分离字符串,得到三个字符串,赋给变量 hour、minute、second,请注意三个变量得到的是字符串值,需要继续做类型转换处理。

```
>>> hour = int(hour)
>>> minute = int(minute)
>>> second = int(second)
>>> hour,minute,second
(7, 30, 25)
```

由于对输入的每一个数据都要进行类型的转换,所以批量数据的输入形式通常使用循环结构控制逐个进行。循环的控制可以是计数型的也可以是按某一约定标识的控制。

【例 5-2-3】 n 个批量数据的输入示例。

实现例 5-1-2 求 n 个数之和的代码如下:

```
n = int(input('n = '))
sum = 0
for i in range(1,n + 1):
    x = int(input())
    sum += x
print('sum = ',sum)
```

运行示例:

```
>>> ================ RESTART ============================
>>>
n=5
56
78
49
31
67
sum= 281
>>>
```

【例 5-2-4】 若干个批量数据的输入示例,输入一批整数计算累加和,输入−1 结束。在本例中,输入一个整数值作为循环控制变量,当输入的值等于−1 时,循环结束,

算法设计如下:

```
1. 累加器置零 sum=0。
2. 输入一个 x。
3. 循环当 x 不等于-1。
   3.1  将 x 累加到 sum。
   3.2  输入一个 x。
4. 显示累加和 sum。
```

在算法第 2 步,x 获取初值,在算法第 3 步判断 x 是否符合循环条件,在第 3.2 步再次输入一个 x,改变循环控制变量 x 的值,然后回到算法第 3 步判断 x 是否符合循环条件,直到 x 的输入值等于−1,循环结束。

实现代码如下:

```python
s=0
x=int(input('请输入一个整数,-1退出:'))
while x!=-1:
    s+=x
    x=int(input('请输入一个整数,-1退出:'))
print('sum=',s)
```

运行示例:

```
>>> ============================ RESTART =========================
>>>
请输入一个整数,-1退出:56
请输入一个整数,-1退出:78
请输入一个整数,-1退出:49
请输入一个整数,-1退出:31
请输入一个整数,-1退出:67
请输入一个整数,-1退出:-1
sum= 281
>>>
```

通常批量数据是存储在列表中的,再支持后续的处理,而例 5-2-5 中只使用一个变量 x,

累加后,变量 x 中的数据被下一个输入数据覆盖不能保留。可以改用列表来存储数据,以支持更复杂的处理。

【例 5-2-5】 输入批量数据到列表,统计它们的个数,总和以及平均值。

算法设计如下:

按输入、计算、输出三大部分安排算法步骤。

```
1. 置空列表 a。
2. 输入一个 x。
3. 循环当 x 不等于 -1。
   3.1  将 x 追加加到列表 a。
   3.2  输入一个 x。
4. 求列表的长度 n。
5. 求列表的累加和 sum。
6. 显示个数、总和和平均值 sum/n。
```

实现代码如下:

```
a=[]
x=float(input('input a number , -1 quit:'))
while x!= -1:
    a.append(x)
    x=float(input('input a number , -1 quit:'))

s=sum(a)
n=len(a)
print('n=',n)
print('sum=',s)
print('average=',s/n)
```

运行示例:

```
>>> ============================ RESTART ==========================
>>>
input a number , -1 quit:56
input a number , -1 quit:78
input a number , -1 quit:49
input a number , -1 quit:31
input a number , -1 quit:67
input a number , -1 quit:-1
n=5
sum=281.0
average=56.2
```

5.2.2 标准输出函数

Python 提供内置函数 print()用于输出显示数据。print()语句的基本格式为

```
print(value, ..., sep='', end='\n', file=sys.stdout, flush=False)
```

参数 value 表示输出对象,可以是变量、常量、字符串等。value 后的“...”表示可以列出多个输出对象,以逗号间隔。

参数 sep 表示多个输出对象显示时的分隔符号,默认值为一个空格。

参数 end 表示 print 语句的结束符号,默认值为换行符,也就是说 print 默认输出后换行。

参数 file 设置输出文件,默认为标准输出即显示器。

参数 flush 设置缓冲,默认为 False。

【例 5-2-6】 输出一个对象示例。

```
>>> print("Welcome!")
Welcome!
>>> x = 100
>>> print(x + 20)
120
```

【例 5-2-7】 输出多个对象示例。

输出三个整数。

```
>>> x,y,z = 10,20,30
>>> print(x,y,z)
10 20 30
```

转换格式输出日期。

```
>>> y,m,d = '2014-1-26'.split('-')
>>> y,m,d
('2014', '1', '26')
>>> print(y,m,d,sep = '/')
2014/1/26
```

【例 5-2-8】 使用 repr 函数构造输出对象示例。

在输出时也要加上友好的用户提示时,可以连续输出字符串对象和数值对象。

```
>>> print('x + 20 = ', x + 20)
x + 20 = 120
```

也可以这样构造一个输出字符串。

```
>>> print('x + 20 = ' + repr(x + 20))
x + 20 = 120
```

运算符+在此表示字符串连接运算,要求符号的两边都是字符串对象,所以使用 repr 函数,将整数对象转换为字符串对象。

也可以通过格式控制符将变量的值按一定的输出格式加入字符串,使用的一般方法为

```
'格式控制串' % (值序列)
```

格式控制串包括普通字符和格式控制符号,普通字符包括所有可以出现在字符串对象中的中英文字符、标点符号、转义字符等,格式控制符号按数据类型如表 5-2-1 所示。

表 5-2-1　格式控制符号

格式符号	表示类型	格式符号	表示类型
%f/%F	浮点数	%o	八进制整数
%d/%i	十进制整数	%x/%X	十六进制整数
%s	字符串	%e/%E	科学计数
%u	十进制整数	%%	输出%

此外还可以加上±m.n的修饰符，m表示输出宽度，n表示小数点位数，+表示右对齐，－表示左对齐。

【例 5-2-9】 使用格式控制符构造输出对象示例。

```
>>> y,m,d = 2014,1,26
>>> '%d-%d-%d'%(y,m,d)
'2014-1-26'
>>> print('%d-%d-%d'%(y,m,d))
2014-1-26
```

【例 5-2-10】 程序设计：一个用户交互友好的求圆面积。

实现代码：

```
import math
radium = float(input("请输入一个圆的半径："))
area = math.pi * math.pow(radium,2)
print('半径为%7.2f 的圆的面积等于%7.2f'%(radium,area))
```

运行示例：

```
>>> ======================== RESTART ==================================
>>>请输入一个圆的半径：12.5
半径为 12.50 的圆的面积等于 490.87
>>>
```

"%7.2f"表示此处需用一个float型数据替换，7表示数据输出占7个字符宽度，2表示输出时小数点保留两位。第一个%7.2f与值序列中的radium相对应，即用变量radium的值替换字符串的格式控制符%7.2f。第二个%7.2f与值序列中的area相对应。

5.2.3 输入输出重定向

通常情况下，Shell环境中的标准输入(STDIN)流，标准输出(STDOUT)流，分别指向键盘、屏幕。系统以文件对象的方式管理外设，Python程序的标准输入输出流定义在sys模块中，分别为sys.stdin，sys.stdout。

输入输出重定向实质是Shell提供的，在控制台窗口中，可以使用重定向符号"<"(输入重定向)和">"(输出重定向)，将输入输出定向到一个文本文件。

例如：在cmd窗口中执行hello.py，如图5-2-1所示。在Windows中运行cmd进入cmd窗口，在命令行提示符后输入cd命令进入文件所在的文件夹，直接执行hello.py，在屏幕下方显示字符串hello，如果使用输出重定向>，再次执行，在屏幕下方不会显示字符串hello，字符串hello写入文件a.txt。

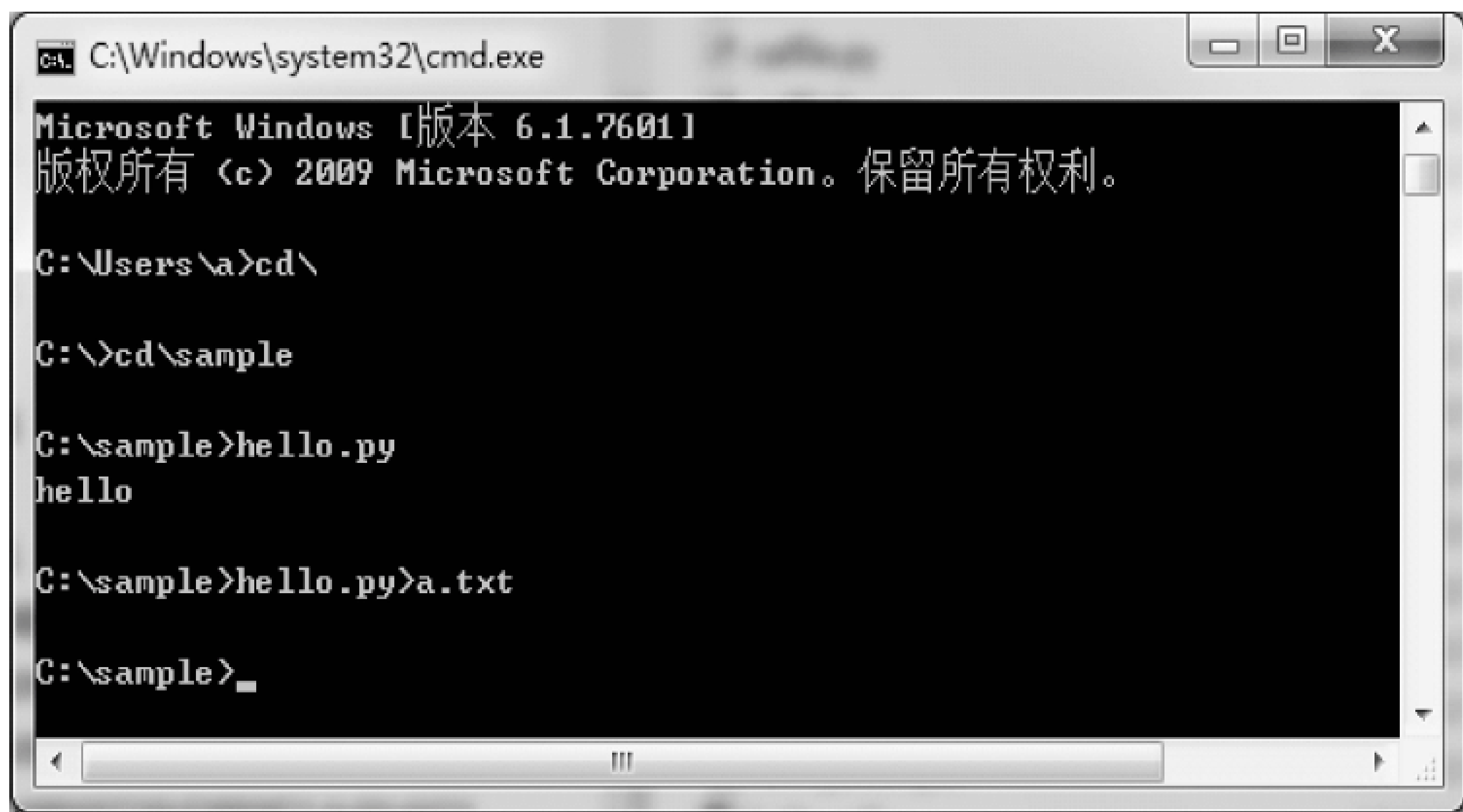


图 5-2-1　输出重定向举例

同样,程序中有 input()语句的,对程序输入重定向到一个文件,程序执行到 input 语句将不从键盘输入数据,而从指定文件中输入数据。

【例 5-2-11】　在指定目录中创建一个名为 input_data. py 的程序,代码如下:

```
a = input()
b = input()
c = input()
print(a,b,c,a + b + c)
```

在同一个目录中建立数据文件 data. txt,内容为

```
10
20
30
```

然后在 cmd 窗口执行 input_data. py,从键盘依次输入三个数据,可看到屏幕显示的数据结果如图 5-2-2 所示。再执行 input_data. py<data. txt。



图 5-2-2　输入重定向举例

5.3 文件输入输出程序

5.3.1 文件的基本操作

Python 的有关文件类的定义在内置模块中，可以直接使用内置函数完成文件的操作。

1. 打开文件

使用 open 函数打开一个文件，返回一个文件对象，函数格式为

```
open(file, mode='r', … )
```

参数 file 表示打开文件的文件名，参数 mode 表示打开文件的方式，默认为 r 表示只读，mode 的符号设置如表 5-3-1 所示。

表 5-3-1　文件打开方式控制字符表

mode	解　　释
r	以只读方式打开
w	以写方式打开一个文件，当这个文件存在时，覆盖原来的内容。当这个文件不存在时，创建这个文件
x	创建一个新文件，以写方式打开，当文件已存在，报错 FileExistsError
a	以写方式打开，写入内容追加在文件的末尾
b	表示二进制文件，添加在其他控制字符后
t	表示文本文件，默认值
+	以修改方式打开，支持读写

最后三个符号 b、t、+是修饰符，添加在 r、w、x、a 的后面，例如 rb 表示以只读方式打开一个二进制文件，r+表示以修改方式打开一个文本文件，rb+表示以修改方式打开一个二进制文件。

例如：创建一个在 C 盘 sample 目录下的 test. txt 文件，返回文件对象 f，f 是由程序员命名的用户标识符，之后就可以使用 f 调用文件操作的方法操作文件。

```
f = open("c:\\sample\\test.txt", "w")
```

说明：

第一个参数是文件名称，包括路径；路径的分隔符\需要转义，用两个\\表示一个路径分隔符。如在当前程序目录下打开，只需写"test. txt"。

第二个参数是打开的模式 mode。解释为以写方式打开的文本文件，如果文件不存在，则自动创建文件。

如果需要以二进制方式打开文件，需要在 mode 后面加上字符"b"，如"rb"、"wb"等。

2. 关闭文件

文件操作完毕，一定要记得关闭文件，可以释放分配给文件的系统资源，供分配给其他文件使用。通过调用文件对象的 close 方法来关闭文件，调用格式为

```
<文件对象>.close()
```

例如，打开一个文件的文件对象为 f，使用结束后关闭文件。

```
f.close()
```

3. 从文件中读取数据

文件类提供了三种方法读取文本文件的内容，分别是：

1) f.read(size)

返回一个字符串，内容为长度为 size 的文本。参数 size 表示读取的数量，可以省略。如果省略 size 参数，则表示读取文件所有内容，作为一个字符串返回。

2) f.readline()

返回一个字符串，内容为文件当前一行的文本。

3) f.readlines()

返回一个列表，列表的数据项为一行的文本[line1，line2，…，line*N*]。再通过循环操作可以逐行访问列表中每一行的内容。

4. 将数据写入文件

```
<文件对象>.write(string)
```

将一个字符串写入文件，write 函数不提供换行，如果需要换行则要通过换行符"\n"。

例如：

```
>>> f.write('hello')
>>> f.write('hello')
```

文件中得到的文本内容是 hellohello。要得到两行的文本可以写为

```
>>> f.write('hello\nhello\n')
```

文件中得到文本：

```
hello
hello
```

5. 文件中的内容定位

f.read()读取之后，文件指针到达文件的末尾，如果再来一次 f.read()将会发现读取的是空内容，如果想再次读取全部内容，必须将定位指针移动到文件开始。

```
f.seek(0)
```

这个函数的格式如下(单位是 Byte)：

```
f.seek(offset, from_what)
```

from_what 值为 0 表示自文件起始处开始，1 表示自当前文件指针位置开始，2 表示自文件末尾开始。参数 from_what * 可以忽略，其默认值为零，此时从文件头开始。文本文件不允许指定 from_what 参数为 1 或 2。必须为非文本文件，其指定的文件打开方式中包含 b，

才能指定任意的 from_what 参数。

offset 表示从 from_what 再移动一定量的距离。

例如：

```
>>> f = open('test.txt', 'w+')
>>> f.write('0123456789abcdef')
>>> f.seek(5)# 定位到文件的第 6 个字节
5
>>> f.read(1)
'5'
>>> f.seek(0)# 定位到文件的开始
0
>>> f.read(1)
'0'
```

5.3.2 文件输入输出程序的实现

1. 从文件中读取批量数值数据

文本文件的纯文本格式特性决定文本文件非常适合作为各种不同应用程序间交流的数据格式，例如可以把一张 Excel 表存储为 CSV 格式的文本文件或直接复制到 txt 文本文件中，在 Python 中读入继续处理。处理后数据再返回 Excel 表。一般应用程序都提供了文本数据的导出导入接口。所以，文本文件是存放批量数据的常用形式。

Python 按文本读入文本文件中的数据，进行处理前，要按数据本身的含义，进行相应的类型转换。

【例 5-3-1】 整数数据读入示例。

如果数据仅仅是一组整数数据，按文件中数据的组织形式可以区分为一行、一列和多行多列，下面按文件中不同组织方式来讨论各种读入数据的方法之间的差异，假设所有的数据文件都存放在 C 盘的 sample 文件夹下。

1）一行

data1.txt 中存放一行整数数据，将其读到一个列表 L1 中。

```
34 78 47 787 84 25 69 25 58 67 52 77 12 67 325 33
```

以只读方式方式打开 data1.txt。

```
>>> f = open("C:\\sample\\data1.txt")
```

使用 read 函数读入全部内容，返回一个字符串对象保存在 *a* 中。

```
>>> a = f.read()
>>> a
'34 78 47 787 84 25 69 25 58 67 52 77 12 67 325 33'
```

通过字符串对象的 split 方法，按空格分离数据，得到一个列表，列表元素为字符串对象。

```
>>> L1 = a.split()
>>> L1
['34', '78', '47', '787', '84', '25', '69', '25', '58', '67', '52', '77', '12', '67', '325', '33']
```

将文件读写位置返回到文件开始。

```
>>> f.seek(0)
0
```

再次使用 readline 方法读一行,返回一个字符串对象,保存在 *a* 中。

```
>>> a = f.readline()
>>> a
'34 78 47 787 84 25 69 25 58 67 52 77 12 67 325 33'
```

如果要得到 L1 列表,方法与上同,调用字符串对象 *a* 的 split 方法即可。

将文件读写位置再次返回到文件开始。

```
>>> f.seek(0)
0
```

使用 readlines 方法按行读取文件,返回一个列表,一行的字符串是列表中的一个元素。

```
>>> a = f.readlines()
>>> a
['34 78 47 787 84 25 69 25 58 67 52 77 12 67 325 33']
```

将列表 *a* 的第一个元素按空格分离到列表 L1 中。

```
>>> L1 = a[0].split()
>>> L1
['34', '78', '47', '787', '84', '25', '69', '25', '58', '67', '52', '77', '12', '67', '325', '33']
>>> f.close()
```

不管用哪一种方法在这里得到的 L1 列表的元素都是字符串对象,应进一步遍历并处理为整数对象,代码可为

```
>>> for i in range(0,len(L1)):
        L1[i] = int(L1[i])
>>> L1
[34, 78, 47, 787, 84, 25, 69, 25, 58, 67, 52, 77, 12, 67, 325, 33]
```

2) 一列

data2.txt 中存放与 data1 相同的整数数据,但以一列的方式存放,将其读到一个列表 L2 中。

以只读方式方式打开 data2.txt。

```
>>> f = open("c:\\sample\\data2.txt")
```

使用 read 函数读入全部内容,返回一个字符串对象保存在 *a* 中。

```
>>> a = f.read()
>>> a
'34\n78\n47\n787\n84\n25\n69\n25\n58\n67\n52\n77\n12\n67\n325\n33'
```

通过字符串对象的 splitlines 方法,按空格分离数据,得到一个列表,列表元素为字符串对象。

```
>>> L2 = a.splitlines()
>>> L2
['34', '78', '47', '787', '84', '25', '69', '25', '58', '67', '52', '77', '12', '67', '325', '33']
```

使用 readlines 方法按行读取文件，返回一个列表，一行的字符串是列表中的一个元素。

```
>>> f.seek(0)
0

>>> L2 = f.readlines()
>>> L2
['34\n', '78\n', '47\n', '787\n', '84\n', '25\n', '69\n', '25\n', '58\n', '67\n', '52\n', '77\n',
'12\n', '67\n', '325\n', '33']
>>> f.close()
```

从上面的运行示例可以看出，readlines 方法适合一列数据的存放格式。read 方法读完数据后需要进行字符串的分离操作。readline 方法每次只能读一行，需要多次执行 readline 函数才能读完 data2 文件，readline 不支持一次性读取数据。

3）多行多列

data3.txt 中存放这 10 行 10 列整数数据，将其读到列表 L3 中。

以只读方式打开 data3.txt。

```
>>> f = open("c:\\sample\\data3.txt")
>>> a = f.read()
```

使用 read 函数读入全部内容，返回一个字符串对象保存在 *a* 中，其中'\t'表示制表位，即一个 Tab 键。

```
>>> a
'679\t69\t191\t510\t73\t815\t182\t583\t96\n153\t450\t109\t107\t443\t371\t140\t136\t242\
n206\t236\t378\t993\t802\t131\t428\t994\t169\n572\t201\t687\t30\t953\t519\t58\t247\t891
\n727\t934\t441\t518\t94\t942\t587\t389\t799\n742\t350\t32\t278\t187\t943\t410\t400\
t286\n466\t475\t962\t123\t541\t682\t140\t691\t439\n154\t560\t600\t535\t156\t162\t77\
t462\t513\n35\t861\t867\t509\t971\t769\t428\t733\t274\n419\t673\t728\t306\t828\t989\
t854\t548\t663\n'
```

split 方法默认按空格键、Tab 键、Enter 键间隔分离数据。

```
>>> L3 = a.split()
>>> L3
['679', '69', '191', '510', '73', '815', '182', '583', '96', '153', '450', '109', '107', '443', '371',
'140', '136', '242', '206', '236', '378', '993', '802', '131', '428', '994', '169', '572', '201', '687',
'30', '953', '519', '58', '247', '891', '727', '934', '441', '518', '94', '942', '587', '389', '799',
'742', '350', '32', '278', '187', '943', '410', '400', '286', '466', '475', '962', '123', '541', '682',
'140', '691', '439', '154', '560', '600', '535', '156', '162', '77', '462', '513', '35', '861', '867',
'509', '971', '769', '428', '733', '274', '419', '673', '728', '306', '828', '989', '854', '548',
'663']
>>> f.close()
```

对于多行多列，使用 readline 和 readlines 方法都需要逐行进行处理，没有 read 方法方便。

针对不同的数据组织方式,可选用不同的方法读入数据,此外,对于只需读入数据的场合,Python 还提供了快速列表访问方式。

```
<列表> = list(open(filename,mode))
```

将一个文件中的数据读入一个列表,一行一个元素,读入的数据类型是字符串对象。不用文件的打开和关闭操作,例如操作 data3.txt 文件,得到列表 L4。

```
>>> L4 = list(open("c:\\sample\\data3.txt"))
>>> L4
['679\t69\t191\t510\t73\t815\t182\t583\t96\n', '153\t450\t109\t107\t443\t371\t140\t136\t242\n', '206\t236\t378\t993\t802\t131\t428\t994\t169\n', '572\t201\t687\t30\t953\t519\t58\t247\t891\n', '727\t934\t441\t518\t94\t942\t587\t389\t799\n', '742\t350\t32\t278\t187\t943\t410\t400\t286\n', '466\t475\t962\t123\t541\t682\t140\t691\t439\n', '154\t560\t600\t535\t156\t162\t77\t462\t513\n', '35\t861\t867\t509\t971\t769\t428\t733\t274\n', '419\t673\t728\t306\t828\t989\t854\t548\t663\n']
```

对列表元素逐个执行分离操作,将分离的结果追加到 L5 列表。

```
>>> L5 = []
>>> for i in L4:
    L5.extend(i.split())
>>> L5
['679', '69', '191', '510', '73', '815', '182', '583', '96', '153', '450', '109', '107', '443', '371', '140', '136', '242', '206', '236', '378', '993', '802', '131', '428', '994', '169', '572', '201', '687', '30', '953', '519', '58', '247', '891', '727', '934', '441', '518', '94', '942', '587', '389', '799', '742', '350', '32', '278', '187', '943', '410', '400', '286', '466', '475', '962', '123', '541', '682', '140', '691', '439', '154', '560', '600', '535', '156', '162', '77', '462', '513', '35', '861', '867','509', '971', '769', '428', '733', '274', '419', '673', '728', '306', '828', '989', '854', '548', '663']
```

最后逐个对列表 L5 的元素做类型转换。

```
>>> for i in range(0,len(L5)):
    L5[i] = int(L5[i])
>>> L5
[679, 69, 191, 510, 73, 815, 182, 583, 96, 153, 450, 109, 107, 443, 371, 140, 136, 242, 206, 236, 378, 993, 802, 131, 428, 994, 169, 572, 201, 687, 30, 953, 519, 58, 247, 891, 727, 934, 441, 518, 94, 942, 587, 389, 799, 742, 350, 32, 278, 187, 943, 410, 400, 286, 466, 475, 962, 123, 541, 682, 140, 691, 439, 154, 560, 600, 535, 156, 162, 77, 462, 513, 35, 861, 867, 509, 971, 769, 428, 733, 274, 419, 673, 728, 306, 828, 989, 854, 548, 663]
```

2. 从文件中读取批量结构数据

所谓的结构数据,也就是一个数据由若干个不同属性的成员构成。例如一户居民一年的用水数据由(户主、户名、表号、上年抄见数、每月抄见数(12 个月的数据))构成,一个可能的记录为

('黄晓明','东川路 156 弄 3 号 504 室','0000359222',772,789,806,847,880,901,950,991,1022,1043,1064,1089,1114),一个文件中保存着多个这样的数据,文件的组织结构一般为每行一户人家的数据。

在程序中表示一户居民的数据可以用一个列表,而表示多户居民应该使用下面形式的

嵌套列表表示：

[[],[],[],[],…,[]]

【例 5-3-2】 结构数据读入示例。

使用列表快速访问方法访问数据文件 data4.txt，文件中存放了 5 户居民一年的用水数据，打开后得到一个 5 个字符串元素的列表。

```
>>> a = list(open("c:\\sample\\data4.txt"))
>>> a
['黄晓明,东川路 156 弄 3 号 504 室,0000359222,772,789,806,847,880,901,950,991,1022,1043,1064,1089,1114\n', '李红,东川路 156 弄 3 号 101 室,0000359201,121,132,145,156,168,179,192,206,219,230,246,258,273\n', '钱多多,东川路 156 弄 3 号 102 室,0000359202,1008,1046,1102,1167,1209,1255,1311,1362,1407,1453,1512,1563,1604\n', '赵志荣,东川路 156 弄 3 号 103 室,0000359203,541,567,590,622,651,689,701,732,758,775,796,814,847\n', '秦天君,东川路 156 弄 3 号 104 室,0000359204,401,412,441,466,479,490,508,522,541,572,603,637,666']
>>>
```

将数据逐个从字符串对象中分离出来，字符串中以“,”分隔数据。以子列表的形式追加到列表 L7 中。

```
>>> L7 = []
>>> for i in a:
    L7.append(i.split(','))
>>> L7
[['黄晓明', '东川路 156 弄 3 号 504 室', '0000359222', '772', '789', '806', '847', '880', '901', '950', '991', '1022', '1043', '1064', '1089', '1114\n'], ['李红', '东川路 156 弄 3 号 101 室', '0000359201', '121', '132', '145', '156', '168', '179', '192', '206', '219', '230', '246', '258', '273\n'], ['钱多多', '东川路 156 弄 3 号 102 室', '0000359202', '1008', '1046', '1102', '1167', '1209', '1255', '1311', '1362', '1407', '1453', '1512', '1563', '1604\n'], ['赵志荣', '东川路 156 弄 3 号 103 室', '0000359203', '541', '567', '590', '622', '651', '689', '701', '732', '758','775', '796', '814', '847\n'], ['秦天君', '东川路 156 弄 3 号 104 室', '0000359204', '401', '412', '441', '466', '479', '490', '508', '522', '541', '572', '603', '637', '666']]
```

遍历嵌套列表元素进行类型转换处理，将 12 个月的用水量转化为整型。

```
>>> for a in L:
        for i in range(3,len(a)):
            a[i] = int(a[i])
>>> L7
[['黄晓明', '东川路 156 弄 3 号 504 室', '0000359222', 772, 789, 806, 847, 880, 901, 950, 991, 1022, 1043, 1064, 1089, 1114], ['李红', '东川路 156 弄 3 号 101 室', '0000359201', 121, 132, 145, 156, 168, 179, 192, 206, 219, 230, 246, 258, 273], ['钱多多', '东川路 156 弄 3 号 102 室', '0000359202', 1008, 1046, 1102, 1167, 1209, 1255, 1311, 1362, 1407, 1453, 1512, 1563, 1604], ['赵志荣', '东川路 156 弄 3 号 103 室', '0000359203', 541, 567, 590, 622, 651, 689, 701, 732, 758, 775, 796, 814, 847], ['秦天君', '东川路 156 弄 3 号 104 室', '0000359204', 401, 412, 441, 466, 479, 490, 508, 522, 541, 572, 603, 637, 666]]
```

3. 输出列表数据到文件

write 函数只能输出一个字符串对象到文件中，数据的分隔、换行等效果都要程序员自行构造字符串完成。

【例 5-3-3】 单层数值数据输出示例。

```
>>> L1 = [34, 78, 47, 787, 84, 25, 69, 25, 58, 67, 52, 77, 12, 67, 325, 33]
>>> f = open('c:\\sample\\data5.txt','w')
>>> for i in L1:
     f.write(repr(i) + '\n')
>>> f.close()
```

在 C 盘的 sample 文件夹中打开 data5.txt 文件，一列排放 L1 中的 16 个整数。读者可以自行实现一行数据，和多行多列数据(例如每行 5 个)的效果。

【例 5-3-4】 嵌套列表数据输出示例。

如果要将例 5-3-2 中的最终结果 L7 输出到文本文件，最好先构造输出字符串，每个居民用水情况列表中前 3 个是字符串，后 13 个是整数，将这 16 个数据连接到一个字符串中，以“,”分隔。

```
>>> f = open('c:\\sample\\data6.txt','w')
>>> s = ''
>>> for a in L7:
     for i in range(0,3):                  # 连接前 3 个字符串
          s = s + a[i] + ','
     for i in range(3,len(a) - 1):         # 连接后 12 个是整数
          s = s + repr(a[i]) + ','
     s = s + repr(a[len(a) - 1]) + '\n'    # 连接最后 1 个整数，加回车符结束
>>> f.write(s)
422
```

>>>f.close()数据文件中的内容如图 5-3-1 所示。

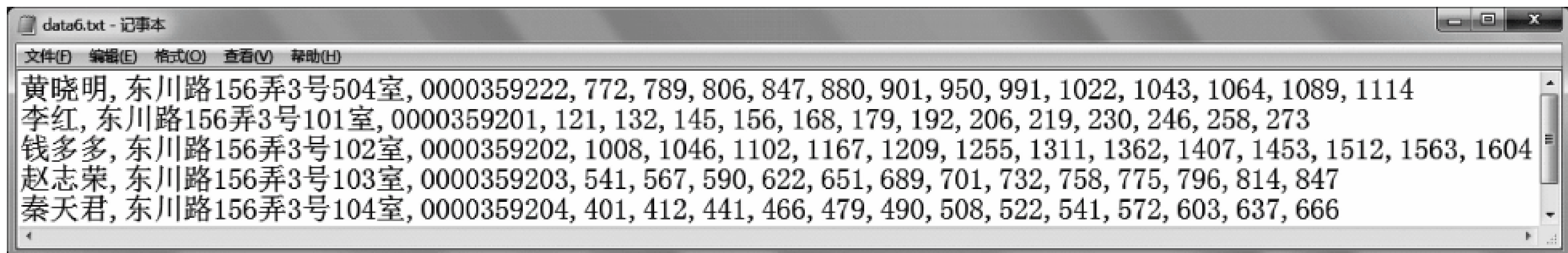


图 5-3-1 批量结构数据输出格式示例

4. 文件输入输出程序综合示例

1) 批量数值数据示例

【例 5-3-5】 程序设计：分析班级计算机课程期末考试情况，统计各个级别的人数。考试成绩按一行存储在文本文件 score.txt 中，级别分类：90 分以上；89～80 分；79～70 分；69～60 分；59～40 分；40 分以下。

算法设计如下：

1. 读入文件数据到列表 L。
   1.1 以只读方式打开文件 score.txt。
   1.2 使用 read 方法读入数据得到一个字符串对象 a。
   1.3 使用 split 方法分离数据得到列表 L，列表中的数据为整数字符串。
   1.4 遍历列表 L 将字符串对象转换为整数。
2. 分类统计各级别人数到列表 c。
   2.1 列表 c 初始化置 0。

2.2　循环迭代遍历 L 中的每一个元素 x。

如果 $x>=90$，则 $c[0]$增 1。

否则如果 $x>=80$，则 $c[1]$增 1。

否则如果 $x>=70$，则 $c[2]$增 1。

否则如果 $x>=60$，则 $c[3]$增 1。

否则如果 $x>=40$，则 $c[4]$增 1。

否则 $c[5]$增 1。

3. 输出各级别统计结果。

程序实现：

```
#读入文件数据到列表 L
f = open("c:\\sample\\score.txt")
a = f.read()
L = a.split()
for i in range(0,len(L)):
    L[i] = int(L[i])
#分类统计各级别人数到列表 c
c = [0,0,0,0,0,0]
for x in L:
    if x>= 90:
        c[0] += 1
    elif x>= 80:
        c[1] += 1
    elif x>= 70:
        c[2] += 1
    elif x>= 60:
        c[3] += 1
    elif x>= 40:
        c[4] += 1
    else:
        c[5] += 1
#输出各级别统计结果
print("90 分以上 %d 人" % c[0],end = ',')
print("89～80 分 %d 人" % c[1],end = ',')
print("79～70 分 %d 人" % c[2],end = ',')
print("69～60 分 %d 人" % c[3],end = ',')
print("59～40 分 %d 人" % c[4],end = ',')
print("40 分以下 %d 人" % c[5],end = '\n')
```

运行结果如下：

```
score.txt 中的数据为
99  63  55  62  13  83  64  92  78  77  84  77  55  97  93  86  82  89  96
97  80  93  69  87  90  84  94  75  76  89  83  83  33  72  48  66  86  98
89  89  88  87  63  87  81  100  80  37  68  71  77  98  66  47  29  87  93
96  100  70  85  83  35
>>> ======================= RESTART ================================
>>>
90 分以上 15 人,89～80 分 22 人,79～70 分 9 人,69～60 分 8 人,59～40 分 4 人,40 分以下 5 人
>>>
```

说明：

① 数据文件中只有一行文件，read 方法和 readline 方法直接得到一个字符串对象，而 readlines 方法和列表快速访问方法都是将这一行的字符串对象作为一个列表元素，故选择 read 方法来完成读入操作。

② 分类统计完成 6 个级别的计数，可选用列表 c 作为计数变量列表，每个列表元素对应一个级别，计数前要置 0。

③ 最后的结果输出可以使用更简单的方法，例如：

直接打印列表：

```
print(c),
输出[15,22,9,8,4,5];
```

或者迭代遍历输出列表元素：

```
for x in c:
    print(x,end = ' ')
输出 15   22   9   8   4
```

但是友好的用户提示是更为重要的设计要素。print 语句中使用了格式控制字符串构建输出提示文字，end 参数可以改变 print 执行后的默认设置，设置为每句以逗号结束。

2）批量结构数据示例

【例 5-3-6】 读取居民用水量数据文件 data4.txt，计算居民每月产生的水费，并将结果输出到结果文件 result.txt 中。2013 年的上海市城镇居民用水计算方法是每立方米 1.630 元，并收排水费每立方米 1.090 元。

程序思路如下：

从文件 data4 中读取批量数据到列表 Ls，Ls 采用嵌套列表结构，一户居民一个子列表，然后逐户逐月计算水费，同样一户居民一个子列表写入列表 Ld 中，最后将列表 Ld 中的数据按一户一行写入文本文件中。

算法设计如下：

1. 读取批量结构数据到列表 Ls。
    1.1　快速列表访问文件 data4.txt，得到列表 t，每行产生一个字符串对象作为列表元素。
    1.2　列表 Ls 置空。
    1.3　循环迭代列表中的每一个元素字符串对象 i。
    　　将 i 按逗号分离得到的列表作为一个元素追加到 Ls 中。
    1.4　循环遍历 Ls 列表。
    　　循环遍历 Ls 列表元素 i 中下标为 3～15 的用水量抄见数(整数字符串)。
    　　将整数字符串转化为整数。
2. 计算水费，结果保存到列表 Ld。
    2.1　列表 Ld 置空。
    2.2　循环迭代读一户居民用水数据到列表 a。
根据列表 a 计算生成一户居民水费数据 d，追加到列表 Ld。
    　　2.2.1　列表 d 置空。

2.2.2 循环遍历列表 *a* 前三个文本数据追加入列表 *d*。
2.2.3 循环遍历列表 *a* 的 1 月份到 12 月份的用水量抄见数(下标: 4～15)。
2.2.3.1 计算该月水费 fee。
2.2.3.2 将水费追加到列表 *d*。
2.2.4 将一户居民的水费记录列表 *d* 追加到列表 Ld
3. 将 Ld 列表数据写入文件 result.txt。
3.1 #以写方式打开文件 result.txt。
3.2 创建输出字符串对象 s。
3.2.1 s 置空。
3.2.2 循环迭代访问一户居民水费记录到列表 *a*。
3.2.2.1 前 3 个字符串数据连接到 s 串,Tab 键分隔。
3.2.2.2 11 个水费浮点数据连接到 s 串,Tab 键分隔。
3.2.2.3 最后一个水费数据连接到 s 串,以 Enter 键结束一行。
3.3 将 s 对象写入文件对象 f。

程序实现代码如下:

```
#读取批量结构数据到 Ls
t = list(open("c:\\sample\\data4.txt"))
Ls = []
for i in t:
    Ls.append(i.split(','))
for i in range(0,len(Ls)):
    for j in range(3,16):
        Ls[i][j] = int(Ls[i][j])
#遍历列表 Ls,计算水费,生成数据到新列表 Ld
Ld = []
for a in Ls:
    #根据列表 a 计算生成一户居民水费数据 d,追加到列表 Ld
    d = []
    for i in range(0,3):
        d.append(a[i])
    for i in range(4,len(a)):
        fee = int(((a[i] - a[i - 1]) * 2.72 + 0.005) * 100)/100    #计算一个月的水费
        d.append(fee)
    Ld.append(d)

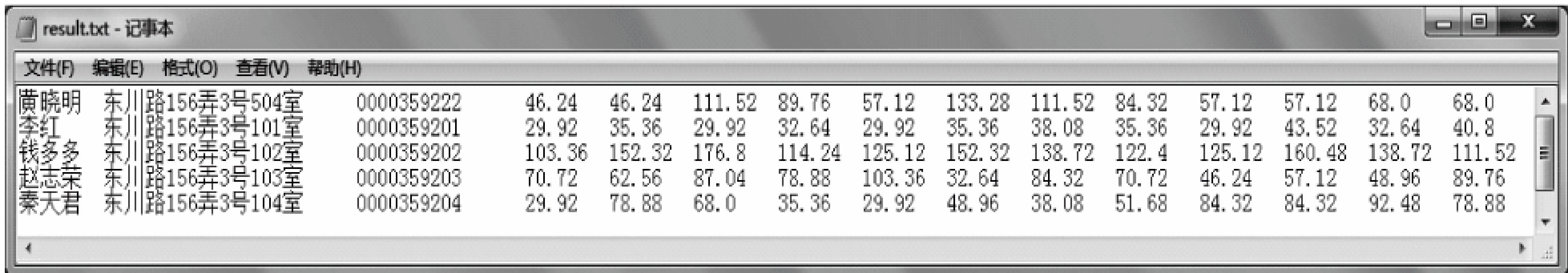
#将 Ld 列表数据写入文件 result.txt
f = open('c:\\sample\\result.txt','w')
#创建输出字符串对象 s
s = ''
for a in Ld:
    for i in range(0,3):
        s = s + a[i] + '\t'
    for i in range(3,len(a) - 1):
        s = s + repr(a[i]) + '\t'
    s = s + repr(a[len(a) - 1]) + '\n'
f.write(s)                                                          #将 s 对象写入文件对象 f
f.close()                                                           #关闭文件
```

运行示例:

生成的 Ld 列表:

```
[['黄晓明', '东川路 156 弄 3 号 504 室', '0000359222', 46.24, 46.24, 111.52, 89.76, 57.12, 133.28,
111.52, 84.32, 57.12, 57.12, 68.0, 68.0], ['李红', '东川路 156 弄 3 号 101 室', '0000359201',
29.92, 35.36, 29.92, 32.64, 29.92, 35.36, 38.08, 35.36, 29.92, 43.52, 32.64, 40.8], ['钱多
多', '东川路 156 弄 3 号 102 室', '0000359202', 103.36, 152.32, 176.8, 114.24, 125.12, 152.32,
138.72, 122.4, 125.12, 160.48, 138.72, 111.52], ['赵志荣', '东川路 156 弄 3 号 103 室',
'0000359203', 70.72, 62.56, 87.04, 78.88, 103.36, 32.64, 84.32, 70.72, 46.24, 57.12, 48.96,
89.76], ['秦天君', '东川路 156 弄 3 号 104 室', '0000359204', 29.92, 78.88, 68.0, 35.36, 29.92,
48.96, 38.08, 51.68, 84.32, 84.32, 92.48, 78.88]]
```

生成的文本文件如图 5-3-2 所示。




```
黄晓明  东川路156弄3号504室  0000359222  46.24  46.24  111.52  89.76  57.12  133.28  111.52  84.32  57.12  57.12  68.0  68.0
李红    东川路156弄3号101室  0000359201  29.92  35.36  29.92  32.64  29.92  35.36  38.08  35.36  29.92  43.52  32.64  40.8
钱多多  东川路156弄3号102室  0000359202  103.36  152.32  176.8  114.24  125.12  152.32  138.72  122.4  125.12  160.48  138.72  111.52
赵志荣  东川路156弄3号103室  0000359203  70.72  62.56  87.04  78.88  103.36  32.64  84.32  70.72  46.24  57.12  48.96  89.76
秦天君  东川路156弄3号104室  0000359204  29.92  78.88  68.0  35.36  29.92  48.96  38.08  51.68  84.32  84.32  92.48  78.88
```

图 5-3-2　生成的居民水费记录文件

5.4　异　　常

5.4.1　简介

在运行程序的过程中,有时会发生些错误,在屏幕上会看到一些非常专业的错误信息,告知错误的名称和发生的原因以及发生错误的程序行,甚至是错误发生时调用的堆栈跟踪情况,然后终止正在运行的程序。这就是非常典型的所谓"错误"或"异常"发生了,而且已经被捕捉到了,并得到了处理——输出相关的出错信息、终止程序的运行。

1. 为什么要使用异常

对于程序设计者来说,发生异常时所看到的这些非常专业化的错误信息是非常有用的,它可以帮助设计者快速定位错误的出处,并改正错误。但对于最终的使用者来说,这些信息就好似天书一般,无法看懂,并且毫无意义。因此,一个好的程序设计者通常要用一种特殊的手段,在程序中设法去捕捉这些发生的错误和异常,并用通俗的、直接面对应用功能的、最终用户能够理解的语言,非常亲切友好地告知用户发生了什么情况(这也是用户喜欢使用这种软件的原因之一),同时做出对异常的相应处理。甚至可以提供多种处理方案让用户选择,用户自己决定应该如何处理。这就是所谓异常处理的主要目的之一。

实际上,程序中可能发生的错误或异常的种类繁多,通常可分为以下三种。

(1) 语法错误(Syntax Error):这是 Python 在对程序脚本做语法解析时所捕捉到并提出警告的错误。例如:Print 的 p 大写了、关键字拼错、括号不配对等。这类错误最容易发现,修改也最为简单,一般在初步调试程序时就会发现并被设计者及时改正,通常不必大费周章地在程序中专门为其编写异常处理代码。这类语法错误,在 Python 中被归类为"错误"(Errors)。

(2) 运行时错误(Run Time Error)：该错误相对于语法错误，不容易被发现。通常是在没有语法错误的情况下发生的，它可能出现在脚本交互的过程中，或者在其他的事件发生时或者某种条件下才会发生，有时甚至是不可避免的。例如：除数为0、用户的非法输入、要打开的文件已经不存在了等。这类错误，在Python中被归类为“异常”(Exceptions)。异常处理最主要的就是针对这种错误。

(3) 逻辑错误(Logical Error)：这类错误是最难发现和清除的，这类错误的代码从语法上来看完全正确，因此程序也会顺利执行，但可怕的是得到的结果却是错误的。例如：在编程中写错了一个变量名，将值错误地赋给了另一个变量、以百分比输出的时候忘记乘以100了等。此类错误无法用异常处理机制捕捉，除非由此导致上述两种错误的出现。

2. 异常的角色

在Python中，虽然异常处理的主要功能是捕捉错误和处理错误，但由于它的特殊机理，还衍生出各种其他的用途。下面是它所担当的最常见的几种角色。

(1) 错误处理(主要角色)：每当在程序运行中检测到程序错误时，Python就会引发异常。可以在程序代码中捕捉和响应错误，或者忽略已发生的异常。如果忽略错误，Python默认的异常处理行为将启动：输出出错信息，终止程序运行(即上文所述情况)。如果不想启动这种默认的异常处理行为，程序员想自己来控制发生异常以后的行为，就要使用专门的语句来捕捉异常并从异常中恢复。

(2) 事件通知：异常也可用于发出有效状态的信号，而不需在程序之间传递结果标志位，或者刻意对其进行测试。

(3) 特殊情况处理：有时，发生了某种很罕见的情况，很难调整代码去处理，通常会在异常处理器中处理这些罕见的情况，从而省去编写应对特殊情况的代码。

(4) 终止行为：可以确保一定会进行某些必需的结束动作的运算，无论程序中是否有异常。

5.4.2 异常处理

一旦程序发生异常，肯定会针对这种异常进行处理。异常处理机制存在于两个地方，一个是程序员写的异常处理代码，另一个就是Python自身的“默认异常处理器”。先来看一下这个所谓“默认异常处理器”的表现。

1. 默认异常处理器

如果程序员没有在脚本代码中刻意编写异常处理代码，或者编写的异常处理代码没有能力捕获或处理所发生的特定的错误，那么这些特定的错误异常一旦发生，就会一直向上传递到程序的最顶层，并启用Python系统自身的“默认异常处理器”：输出标准出错信息，并终止程序的运行。此时，用户也许已经熟悉了标准出错消息，这些消息包括引发的异常名称及说明，还有“堆栈跟踪”信息，也就是异常发生时激活的程序行和程序调用清单。

在Python命令行上输入Print(1)。

```
>>> Print(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

由于 print()函数的 p 大写了,Python 找不到名称为 Print 的函数,错误发生了,系统的默认异常处理器立即启动,给出了错误名称 NameError,告诉用户错误的详细信息:Print 这个名称没有被定义过。其中 stdin 是指出错程序所在的文件名。由于现在处于 Python 命令行状态,输入的语句直接位于标准输入输出设备(屏幕),因此,会看到了一个特殊的设备文件名 stdin。

注意:由于 Python 的 IDLE 工具中外壳行命令执行窗口有时会和真正的执行环境有所不同,它会将下述的 sys. exit()等方法所产生的中断也看作一种异常,会触发它自身的"默认异常处理器"。因此,为了避免混淆,本章节中的所有代码的实现,最好脱离 Python 的 IDLE 工具(毕竟它是一个调试工具),直接在 DOS 窗口中的 Python 中进行测试。

接着,再来看看已经被保存为文件的程序,在运行过程中的错误所引发的默认异常处理器的处理情况。

新建一个简单的程序 Default_exce. py 如下。

```
#Filename:Default_excep
s = input('输入一些东西 --> ')
print('OK! ')
print('你输入的是:' + s)
```

运行该程序时,要在 DOS 窗口中输入 Python Default_excep. py 运行之(或者直接键入文件名 Default_excep. py,省略 Python)。

注意:要使得能找到并启动 Python 虚拟机以及源程序,必须确保 DOS 窗口的当前路径为程序文件所保存的地方,例如:C:\mypython,并同时保证 Python 安装路径已经作为操作系统 path 变量中的一员。

当程序开始运行并等待用户输入时,用户不是老老实实地输入文字或数字信息,而是直接按 Ctrl+Z 键再加上 Enter(Ctrl+Z 键是 Windows 系统中的文件结束符,如果是在 Python 的 IDLE 中运行,应该按 Ctrl+D 键,这来源于 Linux 系统),由此来产生"异常",结果如下:

```
C:\mypython > python Default_excep.py
输入一些东西 --> ^Z
Traceback (most recent call last):
  File "Default_excep.py", line 2, in <module>
    s = input('输入一些东西 --> ')
EOFError
```

此时默认异常处理器的处理结果清楚地告诉用户调用的堆栈跟踪信息:错误出在 Default_excep. py 中的第二行的语句中,错误名是 EOFError。并同时终止了程序(其后的 print 函数没有被执行)。这种处理方式很有道理,因为错误通常都是致命的,而当其发生时,所能做的就是查看标准出错信息,并想法解决问题。

综上所述,默认异常处理器的特征如下。

- 所在地:Python 系统自身的异常处理器。

- 启动时机：程序中的异常处理代码无法捕捉和处理所发生的异常时启动。
- 处理动作：

(1) 终止程序的运行。

(2) 输出标准、专业的出错信息。

① 异常名称及异常说明。

② “堆栈跟踪”信息：异常发生时激活的程序行和程序调用清单。

2. 捕获处理异常 try…except…else…finally 语句

虽然上述默认异常处理器的处理方式和结果能非常好地帮助程序设计者，不过在某些情况下，这并不是我们想要的结果。因为其一程序没错(但却看到了程序错误的提示)，因为程序无法让计算机伸出手臂去阻止用户按 Ctrl+Z 键，只能等错误发生了以后，再做事后的异常处理。其二，最好能给用户一个友好的提示，让用户确切地知道发生了什么事情。其三，或许还能让用户获得重新输入数据的机会。基于以上这些要求，就需要抛开 Python 默认的异常处理器，由自己编写的程序来捕捉异常和处理异常。

异常处理语句的特性如下。

- 所在地：源程序内。
- 启动时机：发生异常时先于默认异常处理器启动，如果已经捕捉到异常，该异常不会再自动被上传至顶层默认异常处理器。
- 处理动作：可按实际应用的需求编写。例如：输出非软件专业的，与实际应用最贴切的出错信息、终止程序运行、完成结尾工作、传递信息、忽略错误继续运行程序，或者重新获得新的数据再次执行前一次出错的程序段等。

1) 异常处理语句 try…except

将上述程序改成如下所示，并另存为 except-input1. py。

```
#Filename:except - input1
import sys
try:
    s = input('输入一些东西 --> ')
except EOFError:
    print('你为何在此按 Ctrl + Z 键啊?')
    sys.exit()                    #退出程序
except:
    print('你到底在干什么啊?')
    sys.exit()                    #退出程序
print('OK!')
print('你输入的是:' + s)
```

运行该程序的结果如下。

```
C:\mypython>python except - input1.py
输入一些东西 --> ^Z
你为何在此按 Ctrl + Z 键啊?
```

这里，我们使用了 try…except 异常捕获处理语句，最重要的就是 try 子句和 except

子句。

- try 子句块：把所有可能引发错误的语句放在 try 子句块中，该子句块中的语句只要发生异常，就会自动将该异常抛出。
- except 子句块：该子句块中的语句通过指定的异常名称，匹配捕捉和处理 try 子句所抛出的错误和异常。一个 except 子句可以专门处理一个或多个错误和异常(要匹配多个异常，必须将多个异常名称用逗号分隔，放在圆括号内)。如果某个 except 子句没有给出任何错误或异常的名称，则它会捕捉和处理剩下的所有错误或异常(好似一个不管部部长捕捉所有没有被匹配和捕捉到的错误)。对于每个 try 语句，至少都要有一个相关联的 except 子句。

try…except 异常处理语句的结构，就好似一个抛球和接球的游戏布局。

上述程序中的 sys. exit()是 sys 模块中的方法，它的作用就是终止程序的执行。

当 try 子句块中的所有语句没有发生任何错误或异常时，立刻跳出 try…except 语句，执行下面的 print('OK!')语句。一旦有错误发生，立刻就去找和该错误名相对应的 except 子句执行，执行完毕后，退出 try…except 语句，继续往下执行(此程序的异常处理结束后没有看到输出的 OK，这是因为在执行 print('OK!')之前，异常处理语句块已经用 sys. exit()方法终止了整个程序的运行)。如果找不到和该错误名相对应的 except 子句，就去找没有附带任何错误名的 except 子句并执行其中的语句块。

try…except 语句在使用时，要注意几个要点：

(1) 不要遗漏子句后的冒号。

(2) try 子句块中可以写多个语句，这些语句执行时所发生的任何错误，都可以用 except 子句捕捉。

(3) 写 except 子句前，必须先确定可能会发生的错误的名称(可以通过实际的错误试验，从默认异常处理器的提示信息中获取)。子句块中的语句，就是针对这些错误的处理语句。

(4) 一句 except 可以捕捉并处理多个指定的错误(在 except 后跟括号，括号中用逗号分开多个错误名称)。

(5) 一句不带错误名的 except 子句，必须放在所有带有明确的错误名的 except 子句后面，它可以捕捉本 try…except 语句内所有前面的 except 子句中没有列出的所有异常(运行上述程序，然后按 Ctrl+C 键看看会有什么结果)。该功能看似万能，但实际使用过程中尽量少用，因为它屏蔽了没有估计到的所有错误和异常，这种 try…except 语句中，默认异常处理器不会再被启动，不利于程序的调试。

(6) 如果所发生的异常找不到相对应的 except 子句，异常就会向上传递到程序先前进入的 try 中(try 嵌套的上层)，或者如果它已经是第一层 try，异常就会被传递到这个进程的顶层(默认异常处理器启动：输出详细的专业出错信息，终止运行程序)。

(7) 一个错误一旦被一句 except 子句捕捉到并处理完后，不会再进入该 try…except 其他的 except 子句，而是直接跳出 try…except 语句，执行 try…except 语句后面的语句。

(8) 如果连 except 子句块中的语句也出错，那么，该错误不会被该 try 语句所捕捉，该异常会上传至上层 try(如果有嵌套的上层 try 语句的话)，如果本 try 语句是第一层，那么异常就会往上传至顶层，启动 Python 的默认异常处理器，输出错误信息，终止整个程序。

上述程序中的错误处理只是输出了出错信息，并立即终止程序。如果对这种处理手法不满意，还可以再次进行调整。例如：如果输入出错了，给出提示让用户重新输入，直到不出错为止。将程序修改后，另存为 except-input2.py，源程序如下：

```
#Filename:except - input2
while True:
    try:
        s = input('输入一些东西 --> ')
        break
    except EOFError:
        print('你为何在此按 Ctrl + Z 键啊?请重新输入!')
print('OK!')
print('你输入的是:' + s)
```

运行该程序，再次使其出错，结果如下：

```
C:\mypython > python except - input2.py
输入一些东西 --> ^Z
你为何在此输入 Ctrl + Z 啊?请重新输入!
输入一些东西 --> ^Z
你为何在此按 Ctrl + Z 键啊?请重新输入!
输入一些东西 --> Thank you!
OK!
你输入的是:Thank you!
```

修改后的程序去掉了屏蔽所有未知错误的单独的 except 子句，以及已经无用的 import sys。加上了一个 while 循环，在 try 子句块中加上了 break 语句，当输入不产生错误时，顺利执行紧挨着的 break 语句，于是跳出循环，执行紧跟在循环后面的语句，输出正确的信息。当输入出现 EOFError 错误时，错误被 except EOFError 子句捕捉，执行 except 子句块，提示错误信息，让用户重新输入，进入下一轮循环，直到输入正确为止。

2) 无异常发生后的 else 子句

接着，再介绍一下 try 语句中另一个可选的 else 子句，它的作用是当 try 子句块中的语句块执行时没有发生任何错误，else 子句中的语句块就会被接着执行，然后退出整个 try 语句，接着执行 try 语句后面的语句。

修改 except-input2.py，另存为 except-input3.py，源程序如下。

```
#Filename:except - input3
while True:
    try:
        s = input('输入一些东西 --> ')
    except EOFError:
        print('你为何在此按 Ctrl + Z 键啊?请重新输入!')
    else:
        break
print('OK!')
print('你输入的是:' + s)
```

该程序执行的效果跟 except-input2.py 一模一样。程序中只是将本来 try 子句块中的 break 语句移入了 else 子句块中。这是一种非常好的习惯,将可能发生异常的代码和无异常发生后必须执行的代码分开来。注意:else 子句只能放在所有 except 子句的后面。它的好处在于:通常在此处将无异常发生的正常情况标志位传递给退出 try 后要执行的语句。它和 try 子句块脱离,在结构和逻辑上清晰易读。

3) 至关重要的 finally 子句

某种情况下,不管 try 子句中的语句是否发生了异常、不管 except 子句是否捕捉到了匹配的异常、是否对异常进行了处理,也不管程序是否会因此被终止,都要完成最后的至关重要的结尾工作。此时,就要用上可选的 finally 子句了。

现有程序 except-finally1.py 如下。

```
# Filename: except - finally1.py
import time
try:
    f = open('poem.txt')
    while True:              # 常用的读取文件的习惯之一
        line = f.readline()
        if len(line) == 0:
            break
        time.sleep(2)
        print(line,end = '')
finally:
    f.close()
    print('\n\n 最终任务已经完成,文件被关闭了!')
print('正常结束!')
```

程序中的 time.sleep()是 time 模块中的延时函数,此处延时 2s。该程序使用循环,一次次读取 poem.txt 文件中的每一行,并以每 2s 一行的速度显示在屏幕上。整个过程中不加人为干预的运行结果如下(必须预先准备好纯文本文件 poem.txt,最好放在与程序相同的路径下)。

```
C:\mypython> python except - finally1.py
当你用你的智慧编写了你自己喜欢的有用的程序,
你会发觉这是一种美好的自我享受。
当你的作品在别人手中争相传递,
你的内心会升腾起无以言表的喜悦!

最终任务已经完成,文件被关闭了!
正常结束!
```

如果你在整个显示过程中按了 Ctrl + C 键(中断程序),此时一个叫做 KeyboardInterrupt 的异常发生了,默认异常处理器启动,程序被终止。但是在程序终止前,finally 子句中最重要的关闭文件动作必须完成。如果不关闭文件,那么这个文件一直处于被打开状态,其他对于文件的操作就会失效,后果很严重。运行结果如下,观察程序被中断

后先前打开的文件是否已被关闭。

```
C:\mypython>python except-finally1.py
当你用你的智慧编写了你自己喜欢的有用的程序,


最终任务已经完成,文件被关闭了!
Traceback(most recent call last):
  File "except-finally1.py", line 9, in <module>
    time.sleep(2)
KeyboardInterrupt
```

通过运行结果看到 finally 子句运行良好,的确关闭了文件。但是,由于没有 except 子句的参与,默认异常处理器依然被启动。现在我们要把 Ctrl+C 也捕捉到,就要增加 except 子句。修改 except-finally1.py,然后将其另存为 except-finally2.py。源程序如下。

```
# Filename: except-finally2.py
import time
import sys
try:
    f = open('poem.txt')
    while True:                # 常用的读取文件的习惯
        line = f.readline()
        if len(line) == 0:
            break
        time.sleep(2)
        print(line,end = '')
except KeyboardInterrupt:
    print('你是如此的没有耐心,按了 Ctrl + C,程序被你中断了!')
    sys.exit()
finally:
    f.close()
    print('\n\n 最终任务已经完成,文件被关闭了!')
print('正常结束!')
```

其中加上了 except KeyboardInterrupt:子句块来捕捉 Ctrl+C 所产生的异常。执行之,并在整个程序执行过程中按 Ctrl+C,结果如下。

```
C:\mypython>python except-finally2.py
当你用你的智慧编写了你自己喜欢的有用的程序,
你会发觉这是一种美好的自我享受.
你是如此的没有耐心,按了 Ctrl + C 键,程序被你中断了!


最终任务已经完成,文件被关闭了!
```

由此看出,异常被捕捉并得到了处理,finally 子句依然正常工作。但此处有一个奇怪的

现象：照理说，应该在 finally 完成工作后跳出 try 语句，继续执行 try 语句下面的其他语句 print('正常结束!')。但在本例中，程序此时已经不可能执行 try 语句下面的其他语句了，因为 sys.exit()退出了程序，看不到"正常结束!"字样了。虽然看上去 finally 在 sys.exit()后执行，但是 Python 为了实现 finally 的设计要求，将程序中断函数自动押后到 finally 以后去执行。综上所述，finally 子句块专门用来做极其重要的扫尾、收拾残局、打扫战场的善后工作，如关闭文件、断开与服务器的连接等。因此为该子句起名为 finally。

另外，由于 Ctrl+C 键是用户手动终止正在运行着的程序的一种常用方法，所以就让它完成它本来的工作——终止程序，不过程序设计者在此时可以提供一些友好的信息。因此，关于 Ctrl+C 键所引起的终止程序动作所引发的 KeyboardInterrupt 异常，通常有两种对待方法。

(1) 不去捕捉它，让默认异常处理器处理：不输出信息，程序被终止。只要做好 finally 子句的设计工作即可。

(2) 去捕捉它，并向用户提供程序被终止的信息，但最终还是以 sys.exit()来终止程序，同样不能遗忘对 finally 子句的设计。

3. 引发(手动抛出)异常 raise 语句

前面所述的所有错误和异常，如果不用 try 语句捕捉，都会引起默认异常处理器的启动，这种错误和异常，可以称之为系统默认异常或内置异常。但在某种情况下，想利用 try 语句的异常处理机制，把一些不会引起默认异常处理器启动的异常，人为地制作成内置异常或用户自定义异常并抛出，然后由 try 语句来捕捉和处理。这样就可以将整个程序中的所有需要处理的异常都设计成在统一机制下的捕捉和处理，便于程序的设计和维护，例如控制用户的数据输入的范围等。

下面是一个接受用户输入年龄，输出用户出生年份的程序 except-raising1.py。

```
# Filename: except-raising1.py
import datetime
class BadAgeException(Exception):
    '''一个用户自定义异常类.'''
    def __init__(self):
        Exception.__init__(self)
        self.message = '你来自未来世界吗?连年龄都是负数 %d ! \
请仔细考虑一下,重新输入!'

while True:
    try:
        yourAge = int(input('输入你的年龄 --> '))
        if yourAge <= 0:
            raise BadAgeException() # 抛出自定义异常类的匿名实例
    except EOFError:
        print('\n 为何给我个 Ctrl + Z 文件结束符? 重新来过!')
    except BadAgeException as x: # 给抛出的自定义匿名实例一个变量名
        print(x.message % yourAge)
    else:
        print('你的年龄是 %d 岁,你生于 %d 年' % (yourAge,
```

```
datetime.date.today().year - yourAge))
            break
    finally:
            print('加油!')
```

该程序用到了 datetime 模块，datetime.date.today().year 的结果是：先从 datetime 该模块中 date 对象的 today 方法得到系统当前的日期对象，再通过该对象的 year 属性得到系统当前日期的年份。

用程序语句来引发一个异常的过程看似比较复杂，下面来看看引发异常所需的准备工作以及引发异常、捕捉处理异常的整个过程。

(1) 程序中预先定义了一个异常类 BadAgeException(用于在发生错误时被实例化并抛出)，它是系统内置异常类 Exception 的子类，并在该类中定义了一个实例变量 self.message，用来定义错误信息(输入的数字小于 0)。实际上，也可以用最简单的方式来定义这个自定义类：class BadAgeException(Exception):pass，然后抛出该类的匿名实例时，传递给构造函数__init__()的参数都会保存在实例的 args 元组属性中，并且在直接打印该实例的时候自动显示。甚至可以传递多个参数，此时可以用实例的 args[0]、args[1]等表示。可参考本章“实验”中的“实验范例”。

(2) try 子句块中判断年龄如果小于 0，就用 raise 抛出一个刚建立的 BadAgeException 类的匿名实例。

(3) 在 try 的 except 子句中捕捉该异常类名，同时也可以用 as x 给抛出的异常类的实例一个变量名字，然后就可以用该实例变量的名字引用自定义异常类中的错误信息了。

当然，由于存在用户的输入，就要将前面所学的对于 Ctrl+Z 的捕捉也编写进去。

运行该程序，输入 20，结果如下。

```
C:\mypython>python except-raising1.py
输入你的年龄 -->20
你的年龄是 20 岁,你生于 1994 年
加油!
```

这是正常的运行结果。接着来试试看输入负数，观察一下引发的异常是否工作正常。输入-9，得到的结果如下。

```
C:\mypython>python except-raising1.py
输入你的年龄 --> -9
你来自未来世界吗?连年龄都是负数 -9 !请仔细考虑一下,重新输入!
加油!
输入你的年龄 -->9
你的年龄是 9 岁,你生于 2005 年
加油!
```

到目前为止，看来一切似乎都很正常。但是，如果输入一些字母，或者小数，会发生什么呢?

```
C:\mypython>python except-raising1.py
输入你的年龄 -->ok
加油!
Traceback (most recent call last):
  File "except-raising1.py", line 12, in <module>
    yourAge = int(input('输入你的年龄 -->'))
ValueError: invalid literal for int() with base 10: 'ok'
```

我们发现在程序中没有与 except 子句匹配的异常发生了，默认异常处理器被启动了。这个错误是：ValueError：invalid literal for int() with base 10：'ok'，它告诉我们 ok 是非法的字面值，不能转换成整型数。于是，要修改程序，必须捕获该异常。

将程序修改后另存为 except-raising2.py，源程序如下。

```
# Filename: except-raising2.py
import datetime
class BadAgeException(Exception):
    '''一个用户自定义异常类.'''
    def __init__(self):
        Exception.__init__(self)
        self.message = '你来自未来世界吗?连年龄都是负数 %d ! \
请仔细考虑一下,重新输入!'

while True:
    try:
        yourAge = int(input('输入你的年龄 -->'))
        if yourAge <= 0:
            raise BadAgeException() #抛出自定义异常类的匿名实例
    except EOFError:
        print('\n 为何给我个 Ctrl+Z 文件结束符? 重新来过!')
    except BadAgeException as x: #给抛出的自定义匿名实例一个变量名
        print(x.message % yourAge)
    except ValueError:
        print('请输入不带小数的阿拉伯数字!重新来过!')
    else:
        print('你的年龄是 %d 岁,你生于 %d 年' % (yourAge,
datetime.date.today().year - yourAge))
        break
    finally:
        print('加油!')
```

运行后依然输入 ok 等错误信息进行测试，结果如下。

```
C:\mypython>python except-raising2.py
输入你的年龄 -->ok
请输入不带小数的阿拉伯数字!重新来过!
加油!
输入你的年龄 -->23.5
```

```
请输入不带小数的阿拉伯数字!重新来过!
加油!
输入你的年龄 --> -12
你来自未来世界吗?连年龄都是负数 -12 !请仔细考虑一下,重新输入!
加油!
输入你的年龄 --> 20
你的年龄是 20 岁,你生于 1994 年
加油!
```

到此为止,大功告成了!

由此可以看到,编程不是一蹴而就的,编程的过程中很大一部分工作就是测试、找出缺陷、修改,再测试……循环往复,不断升级,不断提高,永无止境。

对 Python 中异常处理基本语句的介绍到此告一段落,还有一些更高级的异常处理语句不在本书所涉及的范围内。

5.5 本章小结

本章介绍了数据输入输出的三种方式:标准输入输出、输入输出重定向和文件输入输出与异常处理。

标准输入输出是使用键盘输入数据,结果显示在屏幕上的方式。输入输出重定向是在执行标准输入输出程序时,通过管道命令将输入输出设备改为特定文件的方式。文件输入输出是通过程序提供的文件操作方法访问文件的方式。

异常部分详细介绍了异常是如何发生的、异常的名称、"默认异常处理器"的作用;阐述了在程序中如何控制异常的语句。

本章要点如下:

(1) 程序在接收键盘输入的字符时,要将字符对象转化为程序需要的数据类型,通用的方法是:

```
<类型转化函数>(input([提示字符串]))
```

(2) print 函数负责在将结果输出至屏幕上时,至构造输出的内容时,可以将数据和输出提示作为独立的对象输出,例如:print('x=',x,'y=', y)。print 的 sep 参数可以为输出增加分隔符,end 参数设置 print 函数结束字符,可设置是否换行。

(3) 在输出时,程序通常需要先将不同数据类型的数据加上提示文字来构造输出字符串。构造字符串可以通过字符串连接运算"+",字符串连接运算要求符号的两边都是字符串对象,可以使用 repr 函数将整数对象转换为字符串对象。另外一个方便构造输出字符串的方法是'格式控制串'%(值序列),通过格式控制符将变量的值按一定的输出格式加入字符串。

(4) 访问文件的流程一般为:打开文件、读/写文件和关闭文件。

(5) 通过调用 open 方法打开文件,在文件对象和文件之间建立联系,最后调用 close 方法终止这种联系。打开文件时必须设置操作文件的方式,要改变文件的操作方式则需要关闭文件后,重新打开文件。

(6) 程序设计中最常用的文件形式是文本文件,文本文件由字符数据构成。读取文件

时可以将文件的内容作为一个字符串读出(read 方法),也可以一次读一行(readline 方法),还可以按一行一个字符串,一次读出到一个列表(readlines 方法)。对于只需读入数据的场合,Python 还提供了快速列表访问方式:<列表>=list(open(filename,mode)),就不需要文件的打开和关闭操作了。

(7) 从文件读入的数据都是字符数据,要注意数据类型的转换。

(8) 写文件的实质是操作输出字符串。程序首先构造好输出字符串,将字符串写入文件的方法是 write。与 print 方法不同,write 方法不提供换行。

(9) 文件的读写都是在文件读写的当前位置进行的。打开文件时,如果打开方式为 r、w 或 x,文件的当前位置在文件的开始;如果打开方式为 a,文件的当前位置在文件的末尾。文件的读写操作都会自动改变文件访问的当前位置,seek 方法用于设定文件访问的位置。

(10) 程序中控制异常的语句:try 子句用于捕捉并抛出异常、except 子句通过对异常名称的匹配去处理异常、else 子句将没有异常发生后的处理过程从 try 子句中独立出来、finally 子句用于安排扫尾工作;raise 语句用于在 try 子句中手动触发异常(利用 try 语句的结构来处理系统无法自动识别的错误)。

5.6 习题与思考

5-1 阅读下面的多个变量输入语句,如何输入才能保证变量能正确地获取数据。

(1) `x,y = input("请输入一组坐标值").split(',')`

(2) `a,b,c = input("请输入三个浮点数: ").split()`

(3) 输入若干个单词。

```
L = []
for i in range(1,5):
        L.append(input())
```

5-2 一个处理日期时间的程序已经通过计算得到最后的结果 2014/8/15 13:23:57,数据分别存放在整型变量 *y*(年份),*m*(月份),*d*(日),hh(小时),mm(分钟),ss(秒)中。请写出构造输出字符串 result:2014/8/15 13:23:57 的表达式。

5-3 程序处理学生成绩,处理好后数据存放在列表 L 中,假设该生的数据为[20141356021,雷特,85,67,92,90,76],写出循环访问列表 L 输出的 5 个成绩的语句段,成绩之间用逗号分隔,最后一个数据以句号结束。

5-4 程序处理 *n* 个数的排序问题,处理好后数据存放在列表 L 中。请写出程序段将排序后的数据输出到文件对象 f 中,每行 5 个。

5-5 程序设计,读入一个文件,自动生成一个词汇表,计算每一个词在文件中出现的次数,将词汇表输出到一个文件中保存。注意:单词的大小不同应视为一个单词,例如:let 和 Let 是一个单词,都转化为小写后处理。

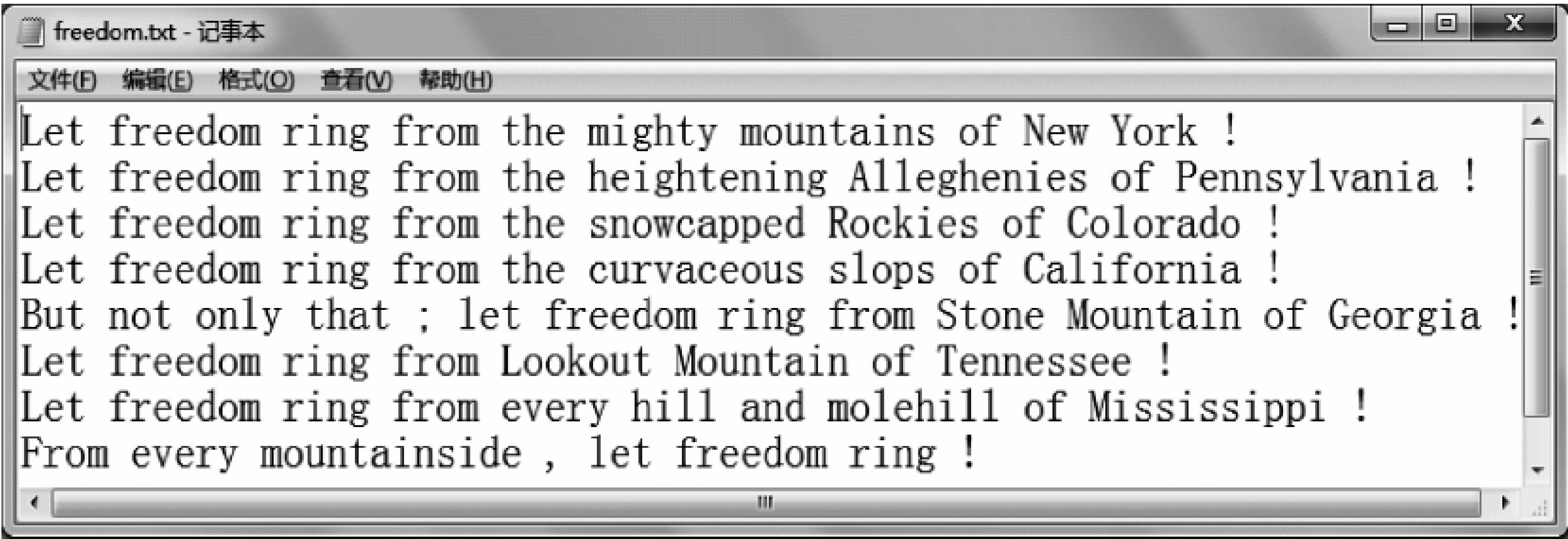
freedom.txt 文件是摘选自马丁路德金的演讲稿 *I have a dream*,如图 5-6-1 所示。

处理后得到单词的使用频率表,如图 5-6-2 所示。

5-6 "默认异常处理器"在何种情况下被触发启动?它是如何处理异常的?

5-7 不带任何错误名的 except 子句该如何使用?

5-8 为何要用 else 子句?

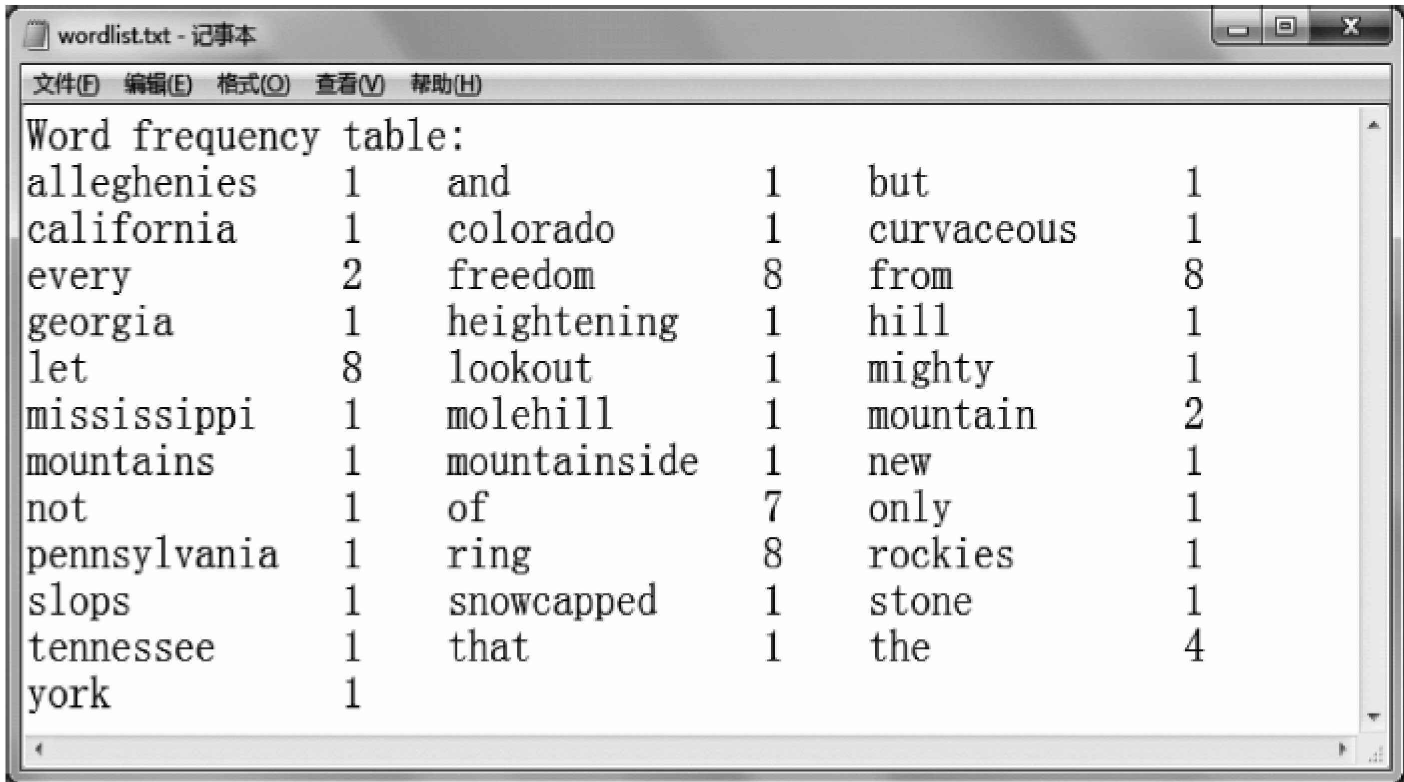


freedom.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Let freedom ring from the mighty mountains of New York !
Let freedom ring from the heightening Alleghenies of Pennsylvania !
Let freedom ring from the snowcapped Rockies of Colorado !
Let freedom ring from the curvaceous slops of California !
But not only that ; let freedom ring from Stone Mountain of Georgia !
Let freedom ring from Lookout Mountain of Tennessee !
Let freedom ring from every hill and molehill of Mississippi !
From every mountainside , let freedom ring !
```

图 5-6-1　原文内容



wordlist.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Word frequency table:

alleghenies	1	and	1	but	1
california	1	colorado	1	curvaceous	1
every	2	freedom	8	from	8
georgia	1	heightening	1	hill	1
let	8	lookout	1	mighty	1
mississippi	1	molehill	1	mountain	2
mountains	1	mountainside	1	new	1
not	1	of	7	only	1
pennsylvania	1	ring	8	rockies	1
slops	1	snowcapped	1	stone	1
tennessee	1	that	1	the	4
york	1				

图 5-6-2　处理后得到的单词频率表

5-9　finally 子句有什么特点？

5-10　raise 语句有什么用途？

5.7　实　　验

5.7.1　标准输入输出

1. 实验目标

(1) 掌握标准输入语句的使用方法，能正确地得到键盘输入的各种类型的数据，并设计合适的用户提示。

(2) 掌握标准输出语句的使用方法，能运用字符串的构造方法，构造友好的用户输出。

(3) 掌握批量数据的内存存储结构的构造方式，能够从键盘读入批量数据到内存存储。

2. 实验范例

(1) 程序设计(5-7-1. py)：使用海伦公式计算三角形面积。a、b、c 为三角形三边边长。

$$\text{area} = \sqrt{s \times (s - a) \times (s - b) \times (s - c)}\ ,\quad s = \frac{(a + b + c)}{2}$$

要求：要考虑不合法的三角形，给出出错提示。

程序分析：

① 三角形的边长和面积为浮点型数据，变量三角形边长 a，b，c，面积 area 和中间变量 s 都为浮点形的变量，浮点型数据输出时需要设计小数点位数。

② 公式计算设计到求平方根需要引入模块 math。

③ 三角形的合法性判断依据是任意两边之和要大于第三边。

算法设计：在例 5-1-1 的基础上，增加对不合法的三角形的判断。

```
1. 输入三角形的三条边长 a、b、c 并转化为浮点型。
2. 如果 a + b<c 或者 a + c<b 或者 b + c<a,
     则输出：输入错误，三角形不合法。
   否则
        2.1  计算 s = (a + b + c)/2。
        2.2  计算面积 area。
        2.3  显示三角形面积。
```

代码实现：5-7-1. py。

```python
import math
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))
if a + b<c or a + c<b or b + c<a:
    print("不是一个合法的三角形!")
else:
    s = (a + b + c)/2
    area = math.sqrt( s * ( s - a) * (s - b) * (s - c) )
    print("area = %8.2f" % area)
```

运行示例：

```
>>> ====================== RESTART ================================
>>>
a = 34.51
b = 61.73
c = 59.36
area =   999.01
>>> ====================== RESTART ================================
>>>
a = 80.24
b = 19.37
c = 154.22
不是一个合法的三角形!
>>>
```

(2) 程序设计(5-7-2. py)：时间计算，输入一个时间，求加 1s 后的时间是多少？

要求：

① 时间采用 24 小时制。

② 设计友好的输入输出提示，表示时间习惯上使用的格式为 h:m:s，输入输出示例如下：

请输入一个时间(h:m:s):15:59:59

加一秒后的时间:16:0:0

程序分析：

题目要求按时间的输入习惯输入，限定了输入的处理为：按一个字符串对象输入后再分离，最后是数据类型转换。输出时也需要做输出字符串的构造。时间加 1s 要考虑到秒、分钟和小时的进位操作：加 1s 后，秒超过了 60，要进位到分钟，分钟超过了 60，要进位到小时，小时超过了 24，要置 0。

算法设计：

```
1. 按 h:m:s 输入时间,按符号:分离到 hour,minute,second。
2. hour,minute,second 转换为整数。
3. second 增 1。
4. 如果 second 达到 60,
     minute 增 1,second 置 0。
5. 如果 minute 达到 60,
     hour 增 1,minute 置 0。
6. 如果 hour 达到 24,
     hour 置 0。
7. 输出更新后的时间。
```

程序代码：

```
hour,minute,second = input('请输入一个时间(h:m:s):').split(':')
hour = int(hour)
minute = int(minute)
second = int(second)

second = second + 1
if second == 60:
    minute += 1
    second = 0
if minute == 60:
    hour += 1
    minute = 0
if hour == 24:
    hour = 0

print('加 1s 后的时间',end = ':')
print('%d:%d:%d'%(hour,minute,second))
```

(3) 程序设计(5-7-3.py)：输入一组实验数据，数据中存在重复数据，计算每一个数据出现的次数。

输入输出示例如下：

请输入一组数据。

```
2.5 3.5 6.5 2.5 2.5 3.5 3.5 3.5 7.5 6.5 6.5 3.5 3.5 7.5 8 2.5 7.5 7 2.5 3.5
(2.50,5) (3.50,7) (6.50,3) (7.00,1) (7.50,3) (8.00,1)
```

程序分析:

考虑到实验数据可以是从不同途径复制得到的,把它作为一行数据,作为一个字符串对象输入,就不用一个一个输入了。字符串对象最终分离并转化为浮点数数据作为列表元素,利用集合的去重复特性可以得到不重复的实验数据,在调用列表的 count 函数分别计算每一个不重复数据出现的次数。构造一个结果列表:

[[实验数据,出现次数],[实验数据,出现次数], ……]

算法设计:

```
1. 从键盘获取实验数据列表 Ls。
    1.1  输入一行实验数据,得到一个字符串对象 a。
    1.2  按空格分离字符串对象 a 到列表 Ls。
    1.3  循环遍历列表对象 Ls[i]。
         将 Ls[i]转化为浮点数。
2. 计算每一个数据出现的次数。
    2.1  获取不重复的实验数据列表 s。
    2.2  将 s 列表按数值从小到大排序。
    2.3  Ld 列表置空。
    2.4  循环遍历 s 中每一个实验数据 x。
         2.4.1  计算 x 在列表 Ls 中出现的次数 c。
         2.4.2  将列表[x,c]追加到列表 Ld 中。
3. 输出结果。
循环遍历 Ld。
    按圆括号输出一组子列表数据,以空格分隔一组数据。
```

代码实现:

```python
#从键盘获取实验数据列表 Ls
print('请输入一组数据')
a = input()
Ls = a.split()
for i in range(0,len(Ls)):
    Ls[i] = float(Ls[i])

#计算每一个数据出现的次数,获得结果列表 Ld
s = list(set(Ls))
s.sort()
Ld = []
for x in s:
    c = Ls.count(x)
    Ld.append([x,c])
#print(Ld)
#输出结果
for x in Ld:
    print('(%.2f,%d)'%(x[0],x[1]),end = ' ')
```

3. 实验内容

(1) 程序设计(5-7-4.py):编写一个程序,配置 0.9%浓度的氯化钠溶液(生理盐水)。输入要配置的溶液数(以升为单位),输出所需要的 10%浓度的氯化钠溶液和注射用水量(以毫升为单位),要求程序有良好的交互提示,输入输出过程如下所示,输出结果小数点保留 1 位。

```
Input L(litre):5
10 % sodium chloride:450.0 ml
Water:4550.0 ml
```

(2) 程序设计:改进范例 5-7-2 中的程序,能够完成下面场合的时间计算。

① 计算加 ns 后的时间(5-7-5.py)。

运行示例:

```
请输入一个时间(h:m:s):12:50:50
输入 ns:800
加 800s 后的时间:13:4:10
```

② 计算加一段时间(h:m:s)后的时间,其中 h 的值为 0~23,m,s 的值为 0~59(5-7-6.py)。

运行示例 1:

```
请输入一个时间(h:m:s):13:50:40
请输入一个时间段(h:m:s):0:25:0
加 0:25:0 后的时间:14:15:40
```

运行示例 2:

```
请输入一个时间(h:m:s):23:35:48
请输入一个时间段(h:m:s):1:25:30
加 1:25:30 后的时间:1:1:18
```

(3) 程序设计(5-7-7.py):编写一个程序处理一组日最高气温。程序需要统计并打印出高温天数(最高温度为华氏 85 或更高),舒适天数(最高温度为华氏 60~85),以及寒冷天数(最高温度小于华氏 60),最后显示平均温度。

用下面数据测试程序:

```
55   62   68   74   59   45   41   58   60   67   65   78   82   88   91   92   90   93   87
80   78   79   72   68   61   59
高温天数:6,舒适天数:14,寒冷天数:6,平均气温:71.2
```

5.7.2 文件输入输出

1. 实验目标

(1) 理解掌握文件的操作函数。

(2) 能按文件的访问流程读入数据和输出数据。

(3) 掌握批量数值数据的内存存储结构构造方式,能够以文件中读入批量数值数据到内存存储。

(4) 了解批量结构数据的内存存储结构构造方式和读取操作。

(5) 掌握批量数据保存到文本文件中的基本方法。

2. 实验范例

(1) 在 Python Shell 环境中操作文本文件。

① 在 C 盘 sample 文件夹中创建 test.txt 文件(sample 文件夹已存在)。

```
>>> f = open("C:\\sample\\test.txt", "w")
```

② 在 test.txt 文件中写入两条记录。

```
>>> f.write("2014-1-21,东方艺术中心,晚 19: 30,维也纳儿童合唱团")
32
>>> f.write("\n2014-6-3,大学生活动中心,晚 18: 00,信息学院毕业晚会")
33
```

返回的 32 和 33 表示写入的字符个数。

③ 读取 test.txt 文件的内容,先要关闭以创建方式打开的文件,再以只读方式打开。

```
>>> f.close()
>>> f = open("C:\\sample\\test.txt", "r")
>>> print(f.read())
2014-1-21,东方艺术中心,晚 19: 30,维也纳儿童合唱团
2014-6-3,大学生活动中心,晚 18: 00,信息学院毕业晚会
```

④ 再次逐条读取记录。

```
>>> f.seek(0)                               #重定位到文件的开头
>>> print(f.readline())
14-1-21,东方艺术中心,晚 19: 30,维也纳儿童合唱团

>>> print(f.readline())
2014-6-3,大学生活动中心,晚 18: 00,信息学院毕业晚会
>>> print(f.readline())
```

⑤ 增加一条记录。以追加方式重新打开 test.txt。

```
>>> f.close()
>>> f = open("C:\\sample\\test.txt", "a")
>>> f.write("2014-6-20,实验 A 楼 213,早 8: 00,计算机考试")
29
>>> f.read()                        #追加方式打开的文件不能做读取操作
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    f.read()
io.UnsupportedOperation: not readable
```

⑥ 以 a+方式打开 test.txt,读取记录。

```
>>> f.close()
>>> f = open("C:\\sample\\test.txt", "a+")
>>> f.read()                        #追加方式打开,文件当前位置在文件的末尾,所以读不到内容
''
>>> f.seek(0)
0
```

```
>>> print(f.read())
2014-1-21,东方艺术中心,晚 19: 30,维也纳儿童合唱团
2014-6-3,大学生活动中心,晚 18: 00,信息学院毕业晚会
2014-6-20,实验 A 楼 213,早 8: 00,计算机考试
>>> f.close()
```

(2) 程序设计(5-7-8.py):修改歌词,文本文件 song.txt 中是歌曲《今天是你的生日我的中国》,请做以下修改,将修改后的歌词显示在屏幕上,并保存在另一个文件中。文件修改效果如图 5-7-1 所示。

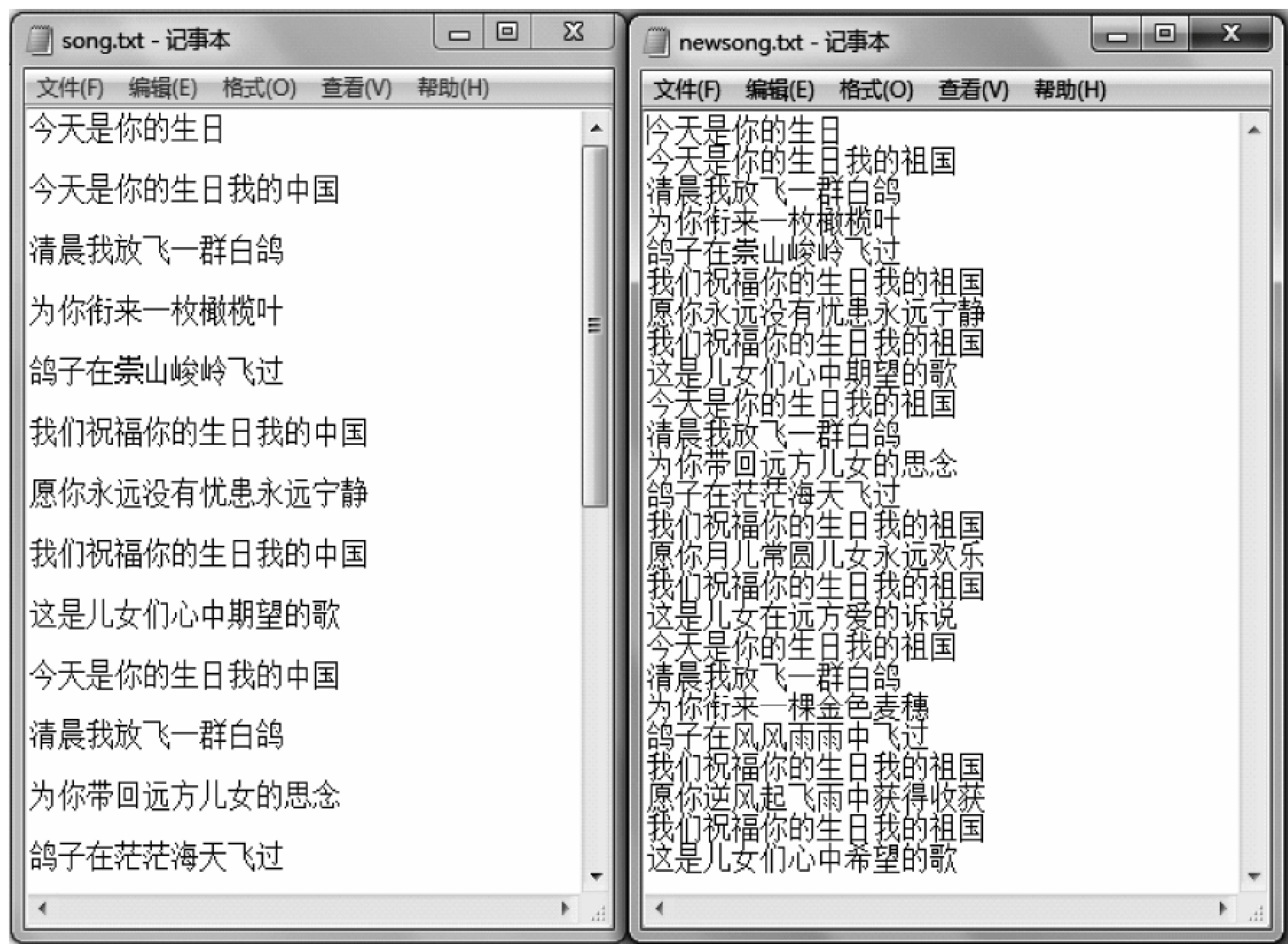


图 5-7-1 歌词文件修改效果对比图

① 每句歌词后一个 Enter 键,删除多余的 Enter 键。

② 将歌词中我的中国改为我的祖国。

程序分析:

使用列表快速访问方法能将 song.txt 文件中一行字符串对象作为列表的一个元素。song.txt 中要删除的多余的空行\n 是列表的一个独立元素。

```
>>> s = list(open('c:\\sample\\song.txt'))
>>> s
['今天是你的生日\n', '\n', '今天是你的生日我的中国\n', '\n', '清晨我放飞一群白鸽\n', '\n', '为你衔来一枚橄榄叶\n', '\n', '鸽子在崇山峻岭飞过\n', '\n', '我们祝福你的生日我的中国\n', '\n', '愿你永远没有忧患永远宁静\n', '\n', '我们祝福你的生日我的中国\n', '\n', '这是儿女们心中期望的歌\n', '\n', '今天是你的生日我的中国\n', '\n', '清晨我放飞一群白鸽\n', '\n', '为你带回远方儿女的思念\n', '\n', '鸽子在茫茫海天飞过\n', '\n', '我们祝福你的生日我的中国\n', '\n', '愿你月儿常圆儿女永远欢乐\n', '\n', '我们祝福你的生日我的中国\n', '\n', '这是儿女在远方爱的诉说\n', '\n', '今天是你的生日我的中国\n', '\n', '清晨我放飞一群白鸽\n', '\n', '为你衔来一棵金色麦穗\n', '\n', '鸽子在风风雨雨中飞过\n', '\n', '我们祝福你的生日我的中国\n', '\n', '愿你逆风起飞雨中获得收获\n', '\n', '我们祝福你的生日我的中国\n', '\n', '这是儿女们心中希望的歌']
```

可以通过列表方法 remove(value)逐个删除'\n'。字符串替换可以逐个使用字符串对象的 replace 方法完成。

算法设计：

```
1. 将文本文件 song.txt 中的歌词逐行读入列表 s。
2. 计算列表中空 Enter 键的个数 n。
3. 循环计数 n 次。
   3.1  删除列表中的一个空 Enter 键。
4. 循环遍历列表每一个 s[i]。
   4.1  将字符串对象 s[i]中的中国替换为祖国。
5. 以写方式打开文件 newsong.txt 返回文件对象 f。
6. 循环迭代遍历列表 s 中的元素 x。
   6.1  将 x 显示在屏幕上。
   6.2  将 x 输出到文件对象 f 中。
7. 关闭文件 f。
```

实现代码：

```
s = list(open('c:\\sample\\song.txt'))
n = s.count('\n')
for i in range(1,n + 1):
    s.remove('\n')
for i in range(0,len(s)):
    s[i] = s[i].replace('中国','祖国')
f = open('c:\\sample\\newsong.txt','w')
for x in s:
print(x,end = '')
    f.write(x)
f.close()
```

思考：如将字符串替换的语句

```
for i in range(0,len(s)):
    s[i] = s[i].replace('中国','祖国')
```

改为循环迭代语句：

```
for x in s:
    x.replace('中国','祖国')
```

能不能达到同样的效果，为什么？

(3) 程序设计(5-7-9.py)：考核评定，某职校对学员进行网络工程师岗位技能测试，测试由网络理论、网络组网实践和网络安全实践三部分成绩构成，考核评定的规定如下：

① 后两门课实践课都达到 60 分，总分达到 180 为合格；

② 每门课达到 80 分，总分 255 分为优秀；

③ 总分不到 180 分或有任意一门实践课不到 60 分则不合格。

学生的考核成绩已经汇总到 grade.txt 文件中，每行一位学员的考核成绩信息，依次为考号、网络理论成绩，网络组网实践成绩和网络安全实践成绩。请对该文件中的学员的考核成绩信息进行评定，给出“优秀”、“合格”、“不合格”的评定结果，按每行一条记录(考号，评定

结果)写到一个新文件中,如图 5-7-2 所示。

```
students.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
10132150105	62	70	70
10132150106	61	80	80
10132150107	65	69	69
10132150108	74	50	50
10132150109	80	28	28
10132150110	80	60	60
10132150111	96	90	90
10132150112	67	85	85
10132150113	78	93	93
10132150114	71	61	61
10132150115	91	52	52
10132150116	75	75	75
10132150117	81	71	71
10132150118	77	90	90
10132150119	63	80	80
10132150120	64	96	96
10132150121	66	93	93
```

```
sturesult.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
10132150105	合格
10132150106	合格
10132150107	合格
10132150108	不合格
10132150109	不合格
10132150110	合格
10132150111	优秀
10132150112	合格
10132150113	合格
10132150114	合格
10132150115	不合格
10132150116	合格
10132150117	合格
10132150118	合格
10132150119	合格
10132150120	合格
10132150121	合格
```

图 5-7-2 学员考核成绩与评定结果对比图

程序分析:

本题思路与例 5-3-6 大致相同,处理批量结构数据,需要嵌套列表[[],[]...]支持结构数据的存储。这样的源数据列表 Ls 和结果数据列表 Ld 中每一个元素还是一个列表 x,子列表 x 还可以通过下标 i 来引用子列表中的元素 $x[i]$。

算法设计:

```
1. 读取批量结构数据到列表 Ls。
   1.1  快速列表访问文件 student.txt,得到列表 s,每行产生一个字符串对象作为列表元素。
   1.2  列表 Ls 置空。
   1.3  循环迭代列表中每一个元素字符串对象 x。
        1.3.1  将 x 按空格分离得到的列表作为一个元素追加到 Ls 中。
   1.4  循环迭代 Ls 列表的每一个元素列表对象 x。
        1.4.1  循环遍历列表元素 x 中下标从 1 开始的成绩数据(整数字符串)。
               1.4.1.1  将整数字符串转化为整数。
2. 判定考核成绩,结果保存到列表 Ld。
   2.1  列表 Ld 置空。
   2.2  创建评定成绩字典{1:'优秀',2:'合格',3:'不合格'}。
   2.3  循环迭代读一个学生成绩信息到列表 x。
        2.3.1  计算总成绩。
        2.3.2  如果每门课达到 80 分,总分 255 分,则 key = 1。
        否则,如果后两门课实践课都达到 60 分,总分达到 180,则 key = 2。
            否则 key = 3。
        2.3.3  将一个学生的考号和评定结构[x[0],d[key]]列表追加到列表 Ld。
3. 将 Ld 列表数据写入文件 sturesult.txt。
   3.1  #以写方式打开文件 result.txt
   3.2  循环迭代访问一个学生的评定记录到列表 x:
        3.2.1  输出一组评定记录到文件对象 f,数据间以 Tab 间隔,以回车结束一行。
   3.3  关闭文件。
```

```
#读取批量结构数据到列表 Ls
s = list(open('c:\\sample\\students.txt'))
Ls = []
for x in s:
    Ls.append(x.split())
for x in Ls:
    for i in range(1,len(x)):
        x[i] = int(x[i])
#判定考核成绩,结果保存到列表 Ld
Ld = []
d = dict({1:'优秀',2:'合格',3:'不合格'})
for x in Ls:
    sum = x[1] + x[2] + x[3]
    if x[1]>= 80 and x[2]>= 80 and x[3]>= 80 and sum >= 255:
        key = 1
    elif x[2]>= 60 and x[3]>= 60 and sum >= 180:
        key = 2
    else:
        key = 3
    Ld.append([x[0],d[key]])
#print(Ld)
#将 Ld 列表数据写入文件 sturesult.txt
f = open('c:\\sample\\sturesult.txt','w')
for x in Ld:
    f.write('%s\t%s\n' % (x[0],x[1]))
f.close()
```

3. 实验内容

(1) 程序设计(5-7-10.py):修改 5.7.1 实验内容第(3)题(5-7-7.py),假设一年的日最高气温数据存放在 temperatures.txt 文件中,每行 10 个,程序需要统计并打印出一年的高温天数(最高温度为华氏 85 或更高),舒适天数(最高温度为华氏 60~85),以及寒冷天数(最高温度小于华氏 60),最后显示平均温度。

运行示例:

```
高温天数:63,舒适天数:162,寒冷天数:140,平均气温:66.7
```

(2) 程序设计(5-7-11.py):已知客户文本文件(customer.txt)有两列数据:姓名和身份证号,计算客户的年龄和性别,按每行一条记录(姓名、性别、年龄)写到新文件中。

要求:

① 请自己查阅身份证编码中提取年龄和性别的方法。

② *根据当前系统日期计算实足年龄,自行查阅 Python 获取系统时间的方法。图 5-7-3 中的实足年龄根据当前日期为 2014-8-15 计算得到的。

(3) 程序设计(5-7-12.py):编写一个程序,实现文件复制的功能,将一个已存在的文件复制到指定目录。

要求:

① 原文件和新文件的名称由程序运行时用户输入。

② 要考虑文件可以在不同的目录。

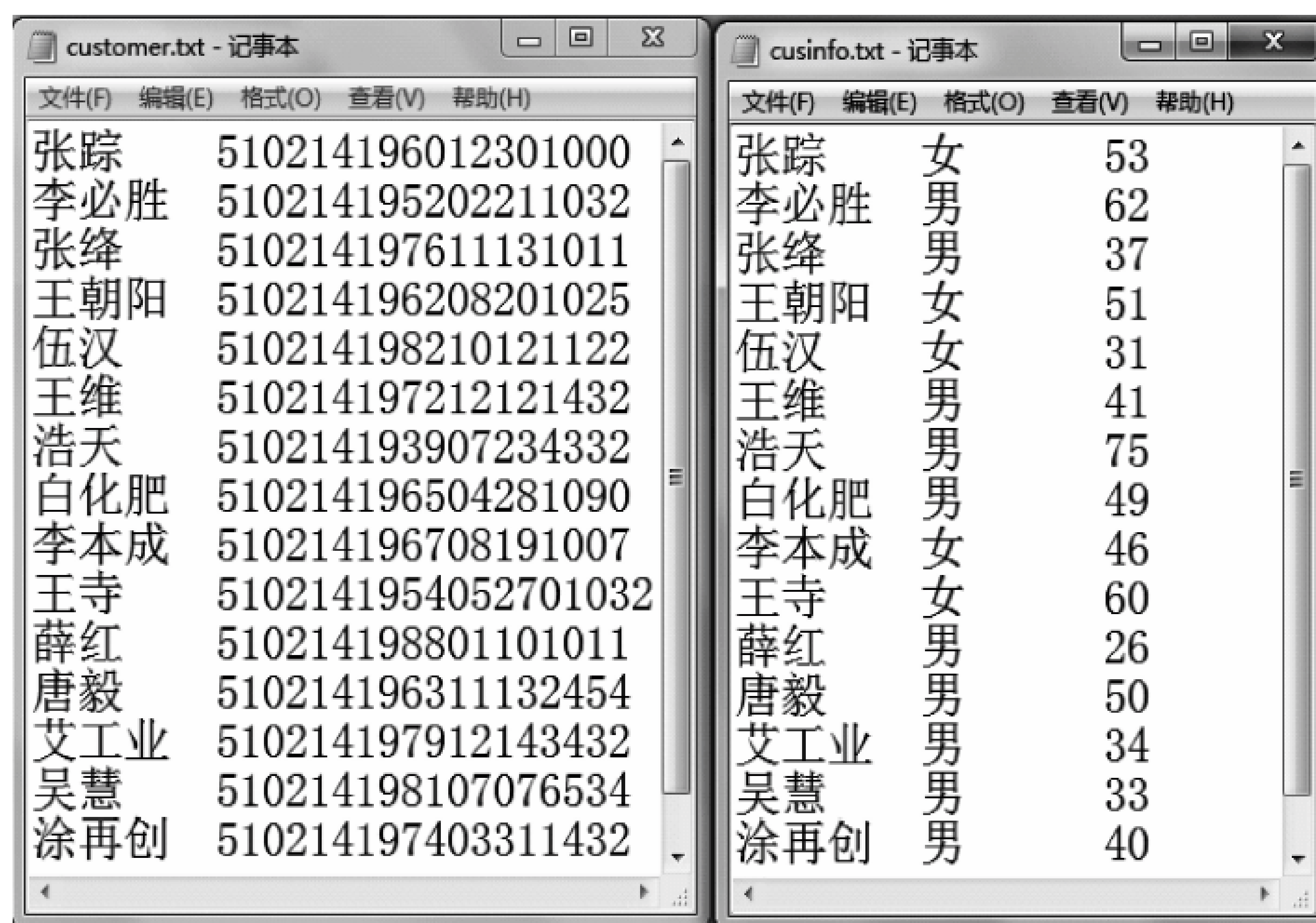


图 5-7-3　客户文件和处理后的结果文件对比

③ 检查原文件如果不存在，给出出错提示。

④ 如新文件已存在，需要给出用户提示，同意后继续，否则放弃复制。

运行示例：

```
>>> ====================== RESTART ================================
>>>
输入原文件名:C:\sample\cusinfo.txt
输入目标文件名: C:\sample\newcusinfo.txt
文件复制成功!
>>> ====================== RESTART ================================
>>>
输入原文件名:C:\smaple\cuseinfo.txt
输入目标文件名: C:\sample\newfile.txt
原文件不存在
>>> ====================== RESTART ================================
>>>
输入原文件名:C:\sample\cusinfo.txt
输入目标文件名: C:\sample\newcusinfo.txt
目标文件已存在,是否覆盖该文件?(Y/N)Y
文件复制成功!
>>>
```

【提示：OS 模块提供了 os.path.exists(文件名)的方法检查文件是否存在。】

(4) 程序设计(5-7-13.py)：编写一个程序，实现写诗机的功能。写诗机的灵感来自于20 世纪 60 年代风靡欧美的小游戏 Madlibs。程序运行时，用户可以按要求输入一些词，将用户输入的词填入一个准备好的文本的空格中，一首新的诗歌就诞生了。用户事先不知道原文是什么，按照所输入的词得到的结果可能是非常有趣的。

例如：包含了哈姆雷特独白的输入文件 hamlet.txt 如下所示，尖括号的内容为用户填空的内容。

```
<verb1>或<verb2>, 这是个问题:
是否应默默地忍受<adjective>命运之无情打击,
还是应与深如大海之<noun>苦难奋然为敌,并将其克服。
此二抉择, 究竟是哪个较崇高?
```

要求编写程序读入此文件,将空格处按用户的输入填空后并显示。下面显示的是一次运行的结果。

```
>>> ======================== RESTART ==================================
>>>
动作 1:program
动作 2:not program
形容词:random
名词:bugs
program 或 not program,这是个问题:
是否应默默地忍受 random 命运之无情打击。
还是应与深如大海之 bugs 苦难奋然为敌,并将其克服。
此二抉择,究竟是哪个较崇高?
```

(5) 程序设计(5-7-14.py):改写实验(4)写诗机,增加其通用性,可以对任意一个文件进行填空,由用户输入原文件名,就可以得到更多不同的诗。

提示:可以将填空的内容也安排在文件中读入,供程序自动处理。原文如图 5-7-4 所示。当然也可以按自己的想法设计文件格式以适应独特的算法设计。

```
3
<adjective1>:填入一个形容街灯的词,后面不加的, 如昏暗,昏黄,幽静,寂寞等
<adjective2>:填入一个形容树的词,后面不加的, 如枯萎,干枯,死亡,健壮等
<author>:作者
«<adjective1>的街灯»
                <author>
<adjective1>的街灯灵感似的闪了一下
我们的视觉把一个男人从墙角揪了出来
他躺在那里
像一棵<adjective2>的冬青树
```

图 5-7-4 原文内容

第一行表示需填空的个数 3,后紧跟三个填空的描述,描述由两部分构成:前面是文中的替换词如<adjective1>,后部分是输入时用户提示,设计以冒号分割。填空描述完后出现的是带空的正文。

运行示例如下:

```
>>> ============================ RESTART ==================================
>>>
输入文件名:treeman.txt
填入一个形容街灯的词,后面不加的, 如昏暗,昏黄,幽静,寂寞等:
炫目
```

```
填入一个形容树的词,后面不加的,如枯萎,干枯,死亡,健壮等:
张牙舞爪
作者:
皓月

«炫目的街灯»
                    皓月
炫目的街灯灵感似的闪了一下
我们的视觉把一个男人从墙角揪了出来
他躺在那里
像一棵张牙舞爪的冬青树
>>>
```

5.7.3 异常处理

1. 实验目标

(1) 熟悉各种系统内置异常的名称。

(2) 能够用 try 语句编写简单的异常处理程序。

(3) 理解引发自定义异常的意义。

2. 实验范例

直接输入表达式,观察、理解正常现象以及异常引发的默认异常处理器的启动。

```
>>> for i in range(1,5)
  File "<stdin>", line 1
    for i in range(1,5)
                      ^
SyntaxError: invalid syntax

>>> for1 i in range(1,5)
  File "<stdin>", line 1
    for1 i in range(1,5)
         ^
SyntaxError: invalid syntax

>>> for1 + i
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'for1' is not defined

>>> 12/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> import math
>>> math.sqrt(2)
1.4142135623730951

>>> math.sqrt(-2)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error

>>> '1' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly

>>> 1 + '1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for + : 'int' and 'str'

>>> str(1) + '2'
'12'

>>> '1' * 3
'111'
>>> 3 * '1'
'111'
>>> '1' * 3.1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non - int of type 'float'

>>> '6'/'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'

>>> a = 5
>>> str(a) + 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> str(a) + '6'
'56'
>>> print(str(a) + '6')
56
>>> '这是一个数字 % d' % a
'这是一个数字 5'
>>> '这是一个数字 % s' % a
'这是一个数字 5'
>>> '这是一个数字 % d' % '5'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: % d format: a number is required, not str

>>> 4 + spam * 3
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined

>>> list = [1,2,3]
>>> list[1]
2
>>> list[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range

>>> f = open('abc.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'

>>> int('20')
20
>>> int('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'
>>> int(3.2)
3
>>> int('3.2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.2'

>>> True + 5
6
>>> true + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined

>>> a = None
>>> a
>>> 1 + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for + : 'int' and 'NoneType'
>>> del a
>>> del b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined

>>> 1000.1 ** 333
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
OverflowError: (34, 'Result too large')

>>> 'a'>'b'
False
>>> 'a'<'b'
True
>>> 1 <'2'
Traceback (most recent call last):
  File "< stdin >", line 1, in < module >
TypeError: unorderable types: int() < str()

>>> a = 1
>>> a++
  File "< stdin >", line 1
    a++
      ^
SyntaxError: invalid syntax
>>> a --
  File "< stdin >", line 1
    a --
      ^
SyntaxError: invalid syntax
>>> a += 1
>>> a
2
```

在程序中使用 try 语句。

(1) 除法练习：编写 except-division. py 程序，让用户输入被除数和除数，输出答案。要求：允许使用浮点数；允许用户按 Ctrl+C 键中断程序，但要提示用户；提示用户所有的非法输入：非 ASCII 数字、除数为 0、按 Ctrl+Z 键；异常处理完后，程序立刻终止。

源程序如下。

```
# Filename: except - division.py
'''除法练习'''
try:
    a = float(input('被除数: '))
    b = float(input('除数: '))
    print(" %f ÷  %f  =  %f" % (a,b,a/b))
except EOFError:
    print('不要没事按 Ctrl + Z! ')
except ZeroDivisionError:
    print('小学没读好?除数能为 0?')
except ValueError:
    print('请使用半角的阿拉伯数字!')
except KeyboardInterrupt:
print('你自己中断了程序!')
```

(2) 求直角三角形的直角边：编写 except-triangle1. py 程序，让用户输入直角边和斜边的长度，输出第二条直角边的长度。

要求：不允许使用浮点数；允许用户按 Ctrl+C 键中断程序，但要提示用户；提示用户

所有的非法输入：非 ASCII 数字、直角边大于斜边、浮点数、按 Ctrl+Z 键，并将这些非法输入所引起的异常用一个 except 子句匹配；异常处理完后，程序立刻终止。

【提示：直角边大于斜边所产生的对负数开根号、输入非 ASCII 正整数这两种错误，都会引发同样的异常——ValueError。】

源程序如下。

```
# Filename: except - triangle1.py
'''求直角三角形的直角边'''
import math
try:
    a = int(input('直角边长度: '))
    c = int(input('斜边长度: '))
    print('直角边长度为 %d,斜边长度为 %d'%(a,c))
    print('第二条直角边长度为: %f'%math.sqrt(c**2 - a**2))
except (ValueError,EOFError):
    print('请使用半角正整数,斜边必须大于直角边,更不要没事按 Ctrl + Z!')
except KeyboardInterrupt:
    print('你自己中断了程序!')
```

上述程序在运行时，可以观察到如果用户输入的是负数，程序也能正常运行，不会引发异常，但显示结果有问题，因为负数是不能被允许的。因此，必须修改。

(3) 修改 except-triangle1. py 源程序，另存为 except-triangle2. py，使其对用户输入的负数引发异常，提醒用户。

```
# Filename: except - triangle2.py
'''求直角三角形的直角边'''
class BadNumber(Exception):pass
import math
try:
    a = int(input('直角边长度: '))
    c = int(input('斜边长度: '))
    if a < 0 or c < 0:
        raise BadNumber('长度能为负数吗?')
    print('直角边长度为 %d,斜边长度为 %d'%(a,c))
    print('第二条直角边长度为: %f'%math.sqrt(c**2 - a**2))
except (ValueError,EOFError):
    print('请使用半角正整数,斜边必须大于直角边,更不要没事按 Ctrl + Z!')
except BadNumber as x:
    print(x)
except KeyboardInterrupt:
    print('你自己中断了程序!')
```

此程序中，使用了 raise 引发异常。程序中对于 KeyboardInterrupt 异常的处理没有使用 sys. exit()，因为按 Ctrl+C 键本身就会结束程序。

3. 实验内容

(1) 编写一文件名为 except-multiple. py 的程序，以完成以下三个表达式的计算。

```
a/(a-b-1)
math.sqrt(a**2-b**2)
a**b
```


要求：a、b 两变量中的数，由用户输入；它们可以是浮点数；a 和 b 都必须大于 20；捕捉 Ctrl+Z；如果有错，让用户继续重新输入；允许用户按 Ctrl+C 键中断程序，但必须输出提示；每次犯错都输出累积的犯错次数，最后成功完成任务时，要是曾经犯过错，还要输出一次犯错次数。可能的运行结果如下。

```
C:\mypython>python except-multiple.py
第一个数: 18
第二个数: 19
第一个数必须大于第二个数!
你犯了 1 次错误!
第一个数: 19
第二个数: 18
每个数必须大于 20!
你犯了 2 次错误!
第一个数: 28
第二个数: 27
除数不能为 0!
你犯了 3 次错误!
第一个数: me
请使用半角的阿拉伯数字!
你犯了 4 次错误!
第一个数: ^Z
不要没事按 Ctrl+Z!
你犯了 5 次错误!
第一个数: 20000
第二个数: 200
20000.000000 ÷ (20000.000000-200.000000-1) = 1.010152
math.sqrt(20000.000000**2 - 200.000000**2) = 19998.999975
你所输入的数字太大了!
你犯了 6 次错误!
第一个数: 28
第二个数: 22
28.000000 ÷ (28.000000-22.000000-1) = 5.600000
math.sqrt(28.000000**2 - 22.000000**2) = 17.320508
28.000000 ** 22.000000 = 68782299287045578092179575799808.000000
你犯了 6 次错误!
任务完成!
```

(2) 有一个纯文本账单文件 bill.txt，其中每一行都是一个浮点数或整数的账目数字，编写程序 except-bill.py 读取账单文件，并输出每笔账目及其累加值。

要求：先用文本编辑器创建账单文件 bill.txt，保存在程序文件所在的同一文件夹中；程序中要读取的账单文件名由用户输入(实际使用时可以读取其他账单文件)，如果所提供的文件不存在则输出提醒信息，让用户重新输入文件名；每一秒钟读一行；每读一行，输出累加公式和当前累加结果；读取时忽略空行；如果读到非数字，则告知用户出错的数据行

号和错误的内容，并退出程序；如果读取所有数据成功，最后再显示一次“总和为……”的信息；在输出数据时，允许用户按 Ctrl+C 键中断程序，但必须给出提示；不管运行结果如何，最终必须关闭已打开的文件并给出提示。

假设当前的 bill.txt 文件内容如下：

```
43.24
235.3
300
8802
123.85
```

正常的运行结果如下：

```
C:\mypython>python except-bill.py
输入账单文件名: bill.txt
0.000000 + 43.240000 = 43.240000
43.240000 + 235.300000 = 278.540000
278.540000 + 300.000000 = 578.540000
578.540000 + 8802.000000 = 9380.540000
9380.540000 + 123.850000 = 9504.390000

打开的 bill.txt 文件被关闭了!
总和为 9504.390000
结束!
```

如果文件名输入错误，则会发生以下情况。

```
C:\mypython>python except-bill.py
输入账单文件名: bill
bill 文件不存在!请重新输入文件名!
输入账单文件名: bill.txt
0.000000 + 43.240000 = 43.240000
43.240000 + 235.300000 = 278.540000
278.540000 + 300.000000 = 578.540000
578.540000 + 8802.000000 = 9380.540000
9380.540000 + 123.850000 = 9504.390000

打开的 bill.txt 文件被关闭了!
总和为 9504.390000
结束!
```

如果在输出时用户等不及，按了 Ctrl+C 键，则结果如下。

```
C:\mypython>python except-bill.py
输入账单文件名: bill.txt
0.000000 + 43.240000 = 43.240000
```

```
43.240000 + 235.300000 = 278.540000
你是如此的没有耐心,按了 Ctrl + C 键,程序被你中断了!

打开的 bill.txt 文件被关闭了!
```

现在创建另一账单文件 bill2.txt,内容如下(包括空行和非数字信息)。

```
43.24
235.3
300

dsfjds
8802
123.85
```

程序运行后,读取该文件,文件中空行将被忽略,字母信息将被视为错误数据,并提出警告,结果如下。

```
C:\mypython>python except-bill.py
输入账单文件名: bill2.txt
0.000000 + 43.240000 = 43.240000
43.240000 + 235.300000 = 278.540000
278.540000 + 300.000000 = 578.540000
bill2.txt 文件中第 5 行含有无效数字[dsfjds],请修改该文件后再来!

打开的 bill2.txt 文件被关闭了!
```

【提示:较为复杂的程序不可能一下子就写成。一般先从最基本最主要的功能入手,如此题中的读取文件中的内容,然后再将一个个次要的功能设计添加进去,如此题中异常处理,添加一个测试一个,然后再进行总体测试。程序设计一定要站在用户的角度考虑问题,不能仅仅从自己使用的角度出发。】

第 6 章　函数与模块

Python 沿袭 C 语言的风格，用函数构建子程序结构，实现程序的模块化。函数是学习程序设计必须要掌握的重要内容。

6.1　函数的基本概念

1. 什么是函数

函数是程序中完成一定功能的一段独立程序代码，可供程序中的其他代码调用。该段独立代码需要有一个名称以供调用，也可接受调用者传递的参数，使处理更加灵活。函数运行后可以有返回值返还调用者。

通俗地说，函数是一种程序构件，是构成大程序的小程序。

2. 为什么要使用函数

(1) 一次定义多次使用，实现“软件重用”。

使用函数可以避免重复代码出现，使程序更精炼。

(2) 功能切割，模块化，结构化。

使用函数不仅可以使程序的结构清晰，更易于阅读和维护，也便于多人合作共同开发程序，并可实现自顶向下、分而治之、逐步求精的结构化程序设计。

(3) 作为一种程序构件，完成特殊的功能。

函数也是实现递归等算法必不可少的工具。

3. 系统函数和用户自定义函数

一般高级语言系统中都提供了自定义函数的语句和方法，供用户针对实际处理的问题创建函数并组装为完整程序。

此外，系统一般也提供一些函数供用户直接调用，以增强系统的运算处理功能，方便用户使用，这些函数称为系统函数。

在第 3 章已经介绍，Python 语言中的系统函数又分 Python 内建函数(如 abs())和标准库函数(如 math. sqrt())，如图 6-1-1 所示。

<table>
<tr><td>Python
built-in functions</td><td rowspan="2">Library
functions
(import)</td></tr>
<tr><td>User defined
functions(def)</td></tr>
</table>

图 6-1-1　Python 语言中的函数类型

内建函数(又称内置函数)是语言的一部分,可直接使用。例如可以直接在 Python 提示符后输入以下函数求一个数的绝对值和四舍五入值。

```
>>> abs(-5)
5
>>> round(3.1415926,2)
3.14
```

此外,各种数据类型转换函数也都是系统的内建函数,可以直接使用。要了解哪些是 Python 的内建函数,可以在 Python 的帮助文档中搜索关键字 built-in function 得到。

与内建函数不同,库函数需要导入后才能使用,并且在使用时要加上库名前缀。在第 3 章已经初步介绍了使用库函数的方法,本章后面在介绍 Python 模块时将作进一步讨论。

本章的重点是如何在 Python 中创建和使用用户自定义函数。

6.2 在 Python 语言中定义和使用函数

6.2.1 函数定义和调用

1. 函数定义

在 Python 中函数定义的格式为

```
def 函数名(参数表):
函数语句块
[return 返回值]
```

其中:

- 函数名应该是 Python 合法的标识符。函数名后紧跟圆括号及其中的参数。
- 参数可以没有(无参函数),也可有多个,多个参数间用逗号分隔。函数的参数需要在调用该函数时用具体的值替换。
- 参数表右括号后面跟冒号。下一行缩进的部分为函数中的语句。
- 在函数中,可通过 return 语句返回函数的值,也可不用 return 语句(或在 return 后不指定值),这样,函数将返回 None——Python 中表示值为“空”的关键字。

【例 6-2-1】 定义一个不带参数的函数。

```
def pr():
    print("**********")
```

【例 6-2-2】 定义求两数最大值的函数。

```
def Max(a,b):
    if a>b:
        return a
    else:
        return b
```

2. 函数调用

在程序中定义的函数可以多次调用,调用函数的形式为:函数名(实际参数表)。

例如，对例 6-2-1 创建的 pr()函数，可直接通过函数名调用：

```
>>> pr()
**********
```

如果定义的函数存在参数，调用时应提供实际参数。所提供的实际参数个数、位置应与函数定义时相对应。如果函数没有参数，也必须使用空括号。

调用函数的方式如下：

- 函数调用可以直接写在一行中作为语句形式出现。
- 可以在表达式中出现(此时一般需要有返回值)。
- 函数调用也可以作为另一个调用函数的实际参数出现。

例如，对于例 6-2-2，可以使用以下语句形式调用：

```
max = Max(35,x)             #将变量 x 和 35 之间大的数赋值给 max
print("Max = ",Max(x,y))    #在屏幕打印出 x 和 y 中大的数
max = Max(Max(x,y),z)       #将 x、y、z 中最大数赋值给 max,这里进行了函数嵌套调用
```

需要指出的是，在 Python 程序中，函数的定义必须在主程序调用语句之前出现，否则运行将出错。

下面再举几个函数定义及调用的实例。

【例 6-2-3】 不带参数的函数。

```
def pr():
    print("**********")
pr() #call function pr()
print("Welcome!")
pr() #call function pr() again
```

运行效果如图 6-2-1 所示。

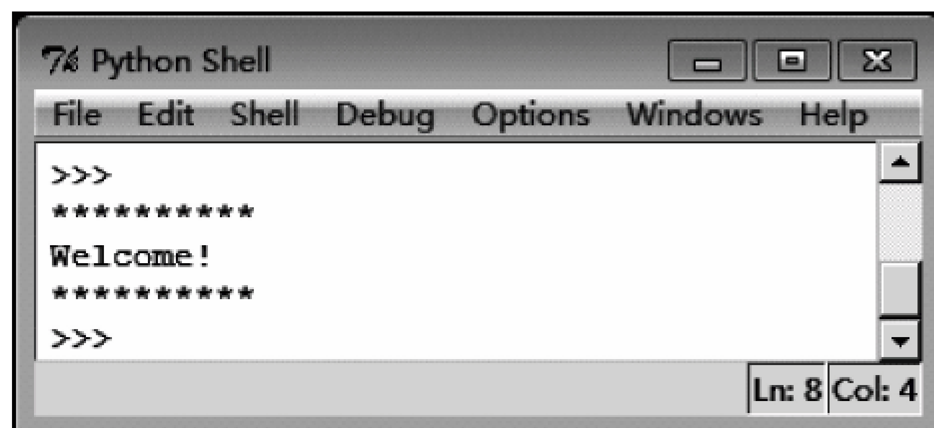


图 6-2-1 例 6-2-3 的运行效果

【例 6-2-4】 带参数的函数。

```
def rt(a):
    for n in range(a):
        for m in range(n+1):
            print('*',end='')
        print()
while True:
    b = input("input Line:")
    if b == '0':
        break
```

```
c = int(b)
rt(c)                          #call function rt()
```

本例通过 rt()函数输出直角三角形图案,在主程序调用该函数时,通过将变量 c 中的数值传递给函数参数 a,给出三角形的行数。

程序运行效果如图 6-2-2 所示。

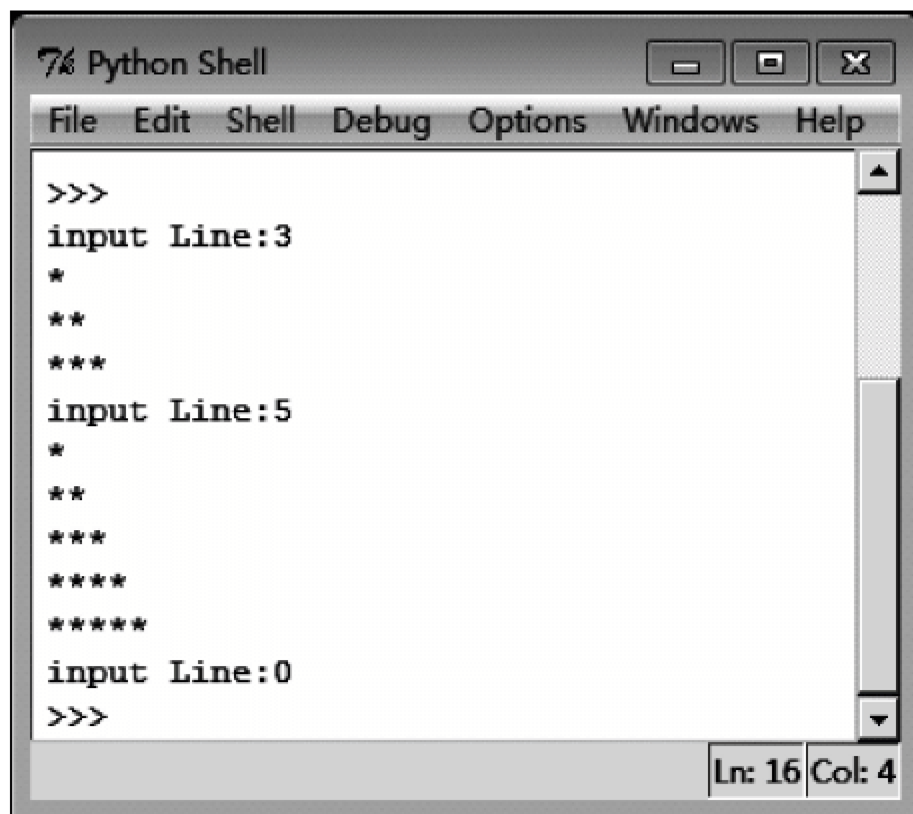


图 6-2-2　例 6-2-4 的运行效果

【例 6-2-5】 带返回值的函数。求阶乘函数:输入 n,计算 $n!$ ($n!=n\times(n-1)\times(n-2)\times\cdots\times2\times1$)。

```
def fact(n):
    factorial = 1
    for counter in range(1,n + 1):
        factorial * = counter
    return factorial
n = int(input("Calculate n! Enter n = ?"))
print(n,'!= ',fact(n))
```

程序运行效果如图 6-2-3 所示。

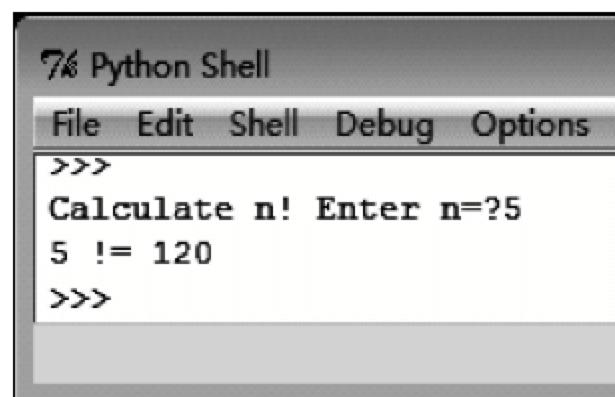


图 6-2-3　例 6-2-5 的运行效果

3. 函数定义与调用中参数间的关系

函数定义时的参数称为形式参数(简称形参),函数调用时的参数称为实际参数(简称实参)。函数调用时所提供的实参个数应与函数定义时的形参一致,两者位置也要互相对应。

例如,对于例 6-2-2 求两数最大值的函数,使用以下语句能正确调用:

```
>>> print(Max(2,3))
3
```

但如果在调用时多给参数或少给参数,均调用失败,如下所示:

```
>>> print(Max(2,3,4))
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
```

```
    print(Max(2,3,4))
TypeError: Max() takes 2 positional arguments but 3 were given
>>> print(Max(2))
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    print(Max(2))
TypeError: Max() missing 1 required positional argument: 'b'
```

一种特殊的情况是，Python 允许在定义函数时提供默认值，即在函数定义时使用：def 函数名(参数 1，参数 2＝值……)：……这种形式。这样，在调用函数时如果没有提供相应的参数，则使用该默认值。

【例 6-2-6】 参数带默认值的函数。

```
def rt1(a = 3):
    for n in range(a):
        for m in range(n + 1):
            print(" * ",end = '')
        print()
rt1()
rt1(5)
```

```
>>>
*
**
***
*
**
***
****
*****
>>>
```

图 6-2-4 例 6-2-6 的运行效果

例 6-2-6 运行效果如图 6-2-4 所示。需要指出的是，如果函数具有多个参数，其中有的参数带默认值，有的没有默认值，则所有没有默认值的参数必须放在前面，最后才为带有默认值的参数，否则程序运行出错。

4. 程序的执行顺序和书写顺序

通过以上内容的学习可知道，Python 程序是由以 def 开头的函数和主程序构成的（这里指非面向对象程序结构）。

1）程序的执行顺序

- 从主程序的入口点语句开始执行，到执行完毕。
- 遇到调用函数，执行转向被调用函数，执行完该函数返回调用处继续向下执行。

2）程序中函数的书写顺序

- 函数的定义必须在主程序调用语句之前出现。
- 多个函数在定义的时候，其书写顺序与其被调用执行的顺序无关。

【例 6-2-7】 以下两程序的运行结果相同。

```
def f2(z):
    y = 1
    print(z + y)
def f1():
    x = 5
    y = 6
    f2(x)
    print(x + y)
f1()
```

```
def f1():
    x = 5
    y = 6
    f2(x)
    print(x + y)
def f2(z):
    y = 1
    print(z + y)
f1()
```

程序均为从最后一条语句（即主程序入口）开始，先调用函数 $f1()$，在执行函数 $f1()$ 中再调用函数 $f2()$。程序的运行结果输出数字 6 和 11。

但如果将最后一条语句（即主程序）放到前面（例如移到第一行），则运行出错。

最后还要指出,函数的定义没有先后之分,也可以如本例所示在一个函数体内调用另一个函数。但要注意,函数是独立的构件,相互之间独立,所以一般不要在一个函数体内定义另一个函数。

6.2.2 函数间的数据联系

1. 局部变量和全局变量

1) 局部变量

在一个函数中使用的变量称为局部变量,不允许在函数外或另一函数中使用。

局部变量是指在函数中定义的变量,一个函数所带的参数也是局部变量。局部变量的作用域只是该函数内部,所以不同的函数中可以有相同名称的变量,它们在各自的函数中互不干扰。当函数执行完毕,局部变量所占有的内存空间也被释放。

【例 6-2-8】 函数中局部变量的作用域演示。

```
def f1():
    x = 5
    y = 6          # f1()中的 y 和 f2()中的 y 互不相干
    print(x + y)
def f2():
    y = 1
    print(x + y) # 出错!不能引用 f1()中的 x
f1()
f2()
```

```
def f1():
    x = 5
    y = 6
    print(x + y)
def f2(x):
    y = 1
    print(x + y)
f1()
f2(5)
```

在左边程序中,由于 x 为 $f1()$ 函数中定义的局部变量,将它用于 $f2()$ 函数程序报错。

如果在该程序里,在 $f2()$ 中设置一个函数参数 x,如右边程序所示,则程序能顺利运行。此时,在两个函数中的 x 为不同的变量,尽管名字相同,但由于分属不同的函数,它们互不干扰。

2) 全局变量

在所有函数外定义的变量为全局变量。全局变量既可用在主程序中,也可以在各函数中使用。

在例 6-2-8 中,如果将变量 x 移到函数外定义——即作为全局变量,如右方程序所示,则可以顺利运行。此时,整个程序中的 x 为同一个全局变量。

```
x = 5
def f1():
    y = 6
    print(x + y)
def f2():
    y = 1
    print(x + y)
f1()
f2()
```

初看起来使用全局变量比较方便,但程序中过多使用全局变量,将使函数间的耦合变得紧密,破坏函数的独立性。

请大家仔细分析这几段程序，体会局部变量和全局变量的不同作用域。

3）局部变量与全局变量的关系

在例 6-2-5 求阶乘函数中，主程序使用全局变量 n 接收用户的键盘输入，并将值传递给函数，在函数 fact(n)中所带的参数 n 是局部变量，这两者在一个函数中的关系遵循以下原则：

如果在函数中定义的局部变量与全局变量同名，则局部变量屏蔽全局变量。

【例 6-2-9】 函数内部局部变量屏蔽同名全局变量的演示。

```
x = 'outside'
y = 'global'
def f():
    x = 'inside'
    print(x)
    print(y)
f()
print(x)
```

程序运行结果为：

```
>>>
inside
global
outside
>>>
```

本例在函数 $f()$中输出的是局部变量 x 的值 inside，在主程序中输出的为全局变量 x 的值 outside。

思考： 例 6-2-5 求阶乘程序，如果直接使用全局变量 n 计算 $n!$，如何修改相应的函数和主程序？

4）使用 global 语句声明全局变量

如果对例 6-2-9 的程序做以下修改，即在函数 $f()$中对全局变量 y 进行赋值运算，则程序运行出错。提示：UnboundLocalError：local variable 'y' referenced before assignment。

```
x = 'outside'
y = 'global'
def f():
    y = y + 'variable'        #运行时报错
    print(x)
    print(y)
f()
```

其原因在于：由于 Python 语言采用“动态类型”技术，变量使用前不需要先声明，对变量的赋值语句即可自动创建变量。所以该程序中的赋值语句 $y=y+$'variable'将在函数内部创建一个局部变量 y，但在创建时等号右方的 y（也是局部变量）还不存在，所以系统提示出错。

为了告知函数某变量为全局变量，可以使用 global 语句，如下所示：

```
x = 'outside'
y = 'global'
def f():
    global y          # 声明 y 为全局变量
    y = y + ' variable'
    print(x)
    print(y)
f()
```

程序运行结果为：

```
>>>
outside
global variable
>>>
```

2. 函数与调用者之间的数据沟通

前已述及，程序中过多使用全局变量，将使函数间的耦合变得紧密，破坏函数的独立性。那么，不使用全局变量如何将一个函数中的局部变量数据传递到外面呢？

在一个函数中使用的局部变量若要与函数外沟通，可以通过以下两种方法：

- 通过参量从调用者输入值；
- 通过返回值向调用者输出值。

例如，下面程序中函数 *f*2()获知函数 *f*1()中某个变量的途径是：①该变量被作为参数传递给 *f*2()；②该变量作为返回值传递。

【例 6-2-10】 将一个函数的内部值传递到函数外。

```
# f1()中的变量 x 作为参数传递给 f2()
def f1():
    x = 5
    y = 6
    f2(x)
    print(x + y)
def f2(z):
    y = 1
    print(z + y)
f1()
```

```
# f1()中的变量 x 作为返回值传递给 f2()
def f1():
    x = 5
    y = 6
    print(x + y)
    return x
def f2():
    y = 1
    print(f1() + y)
f2()
```

以上左边程序，在函数 *f*1()中调用 *f*2()函数，同时将局部变量 x 的值传递到 *f*2()中。右边程序，在函数 *f*1()中将局部变量 x 的值作为函数的返回值传递到调用它的 *f*2()中。

以上两程序，左边运行时输出 6，11，右边运行时输出 11，6。读者可根据程序运行结果，分析各语句的执行顺序，有助于进一步理解函数的调用执行过程。

3. 函数参数的传值和传地址

如上所述，在调用一个函数时，可以通过其调用语句的函数参数，将变量的值传递到所调用的函数中。

调用函数时所提供的实际参数如果是一般变量，仅是单向向函数提供值，在函数中进行的修改不会影响函数外的该变量值。

但列表除外：如果将列表对象作为函数的参数，则向函数中传递的是列表的引用地址。

这时，在函数中对它的操作将直接改变函数外该列表的值。

【例 6-2-11】 传值和传地址演示。

在函数调用时传值：

```
def swap(x,y):
    x,y = y,x
    print('x = ',x,'y = ',y)
a = 5
b = 9
swap(a,b)
print('a = ',a,'b = ',b)
```

程序运行结果为：

```
>>>
x= 9 y= 5
a= 5 b= 9
>>>
```

在函数调用时传地址：

```
def swap(p):
    p[0],p[1] = p[1],p[0]
    print('p0 = ',p[0],'p1 = ',p[1])
a = [5,9]
swap(a)
print('a0 = ',a[0],'a1 = ',a[1])
```

程序运行结果为：

```
>>>
p0= 9 p1= 5
a0= 9 a1= 5
>>>
```

以上第一个程序中，函数 swap(x,y)中的参数为非列表对象，所以在调用该函数时，主程序中 a 的值传给 x，b 的值传给 y，在函数中通过交换赋值，将 x、y 的值进行对换，但主程序中打印输出的 a、b 变量的值并没有交换。

在第二个程序中，函数 swap(p)中的参数为列表对象，当主程序调用该函数时，将列表对象(变量 a)的地址传递给 p，即 p 和 a 指向的是同一处，故当 $p[0]$和 $p[1]$数据交换时，也即使 $a[0]$和 $a[1]$数据的变化。初学者特别要注意这点。

需要指出的是，如果将列表中的部分元素作为函数参数，例如上面定义的函数为 def swap(a[0],a[1])，这种情况等同于一般变量的实参传递。

【例 6-2-12】 在函数中使用列表参数。

```
def insertList(L2,x):
    if x > L2[len(L2) - 1]:          #如果比列表的最后一个元素还要大
        L2.append(x)                 #将要插入的数添加到列表的最后
```

```
        return
    for i in range(0,len(L2)):              # 扫描当前列表
        if x < L2[i]:                       # 如果要插入的数比当前扫描到的列表元素小
            L2.insert(i,x)                  # 则将该数插入当前列表元素之前
            Break                           # 跳出循环
    return
L1 = [1,4,6,9,13,16,28,40,100]
print('初始列表: ',L1)
x = int(input('请输入一个要插入的整数: '))
insertList(L1,x)
print('插入 % d 后: ' % x,L1)
```

本程序的功能是将一个从键盘接收的整数插入列表中。假设该列表已按从小到大的顺序排好了序。函数中对数据的添加和插入直接使用了列表的方法。

由于函数第一个参数传递的是列表,对 L2 的操作也就是对 L1 的操作。程序运行效果如下所示(输入的数为 23)。

```
>>>
初始列表: [1, 4, 6, 9, 13, 16, 28, 40, 100]
请输入一个要插入的整数: 23
插入 23 后: [1, 4, 6, 9, 13, 16, 23, 28, 40, 100]
>>>
```

顺便指出,对于该类程序,因为 L1 和 L2 代表同一个列表对象,将它们用同一个标识符(例如 L)命名即可。

6.3 函数应用

1. 打印图案

【例 6-3-1】 构造函数模块结构打印如图 6-3-1 所示的图形。

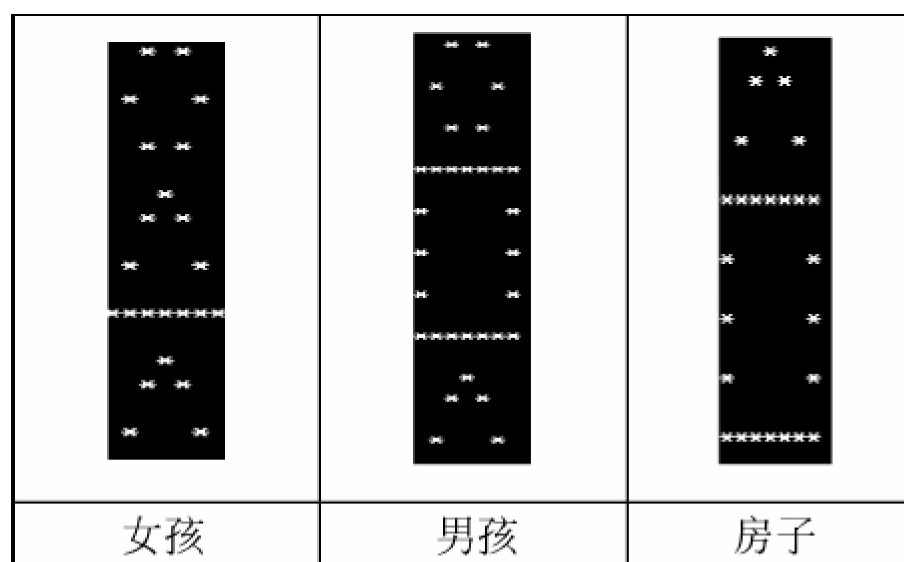


图 6-3-1 待打印的图形

在程序设计中可以对要实现的功能做适当的分解。例如,要打印如图 6-3-1 所示的三种图形,一种方法是一个个单独地把图形打出来。但能否找出这些图形的共同规律,以更高效的方法绘制这些图形呢?

1) 分析

首先从图中可以找到三个图形中一些共同的组件,如男孩和女孩的头都是用圆形实现的,男孩的身体和房子的主体都是用矩形实现的,女孩的腿、男孩的腿及房子的顶都是用交

叉线(∧)实现的。

再进一步观察图形中每一行的实现,有两种类型:一种是点线,线的一头一尾有两个星号,中间是空白;另一种是由连续星号构成的线段。Python 中的 print 语句提供了打印重复字符的方法:字符 * 整数 n,表示输出 n 个相同的字符。例如 print("+" * 4)表示打印连续 4 个加号,print(" " * (start-1))表示打印 start−1 个连续空格。

通过调用以上这两种类型的线形(点线和连续线段)可以构造画圆、矩形和交叉线的函数模块。表示女孩身体的三角形也可以由画交叉线函数加上绘制连续线段构成。

画女孩的过程如图 6-3-2 所示。画女孩可以由画圆、画三角形、画交叉线三部分构成,而这三部分又都由画点线和画连续线段两部分构成。

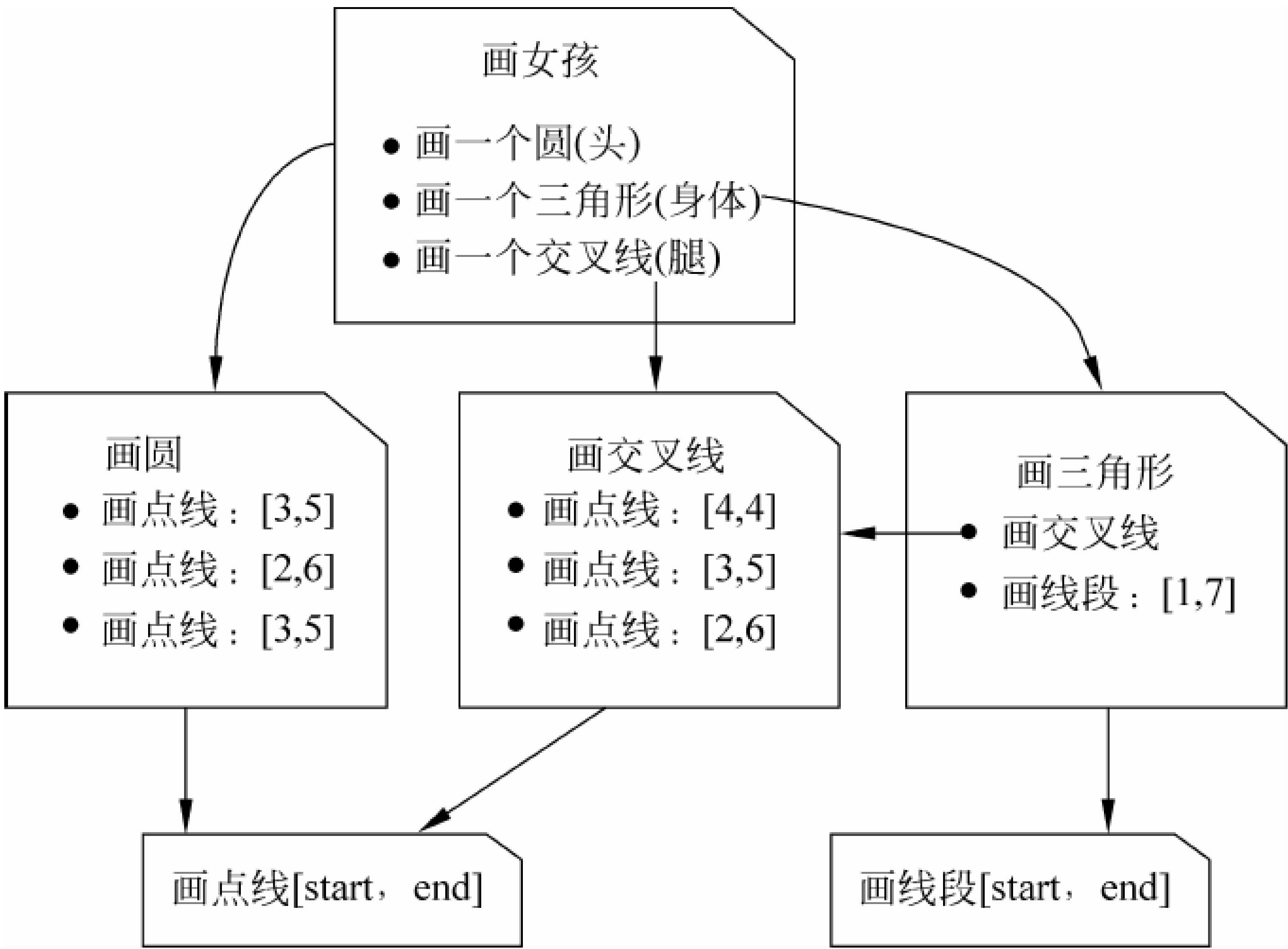


图 6-3-2 画女孩的过程

通过上面的分析,可以得到如图 6-3-3 所示的画女孩的模块结构图。

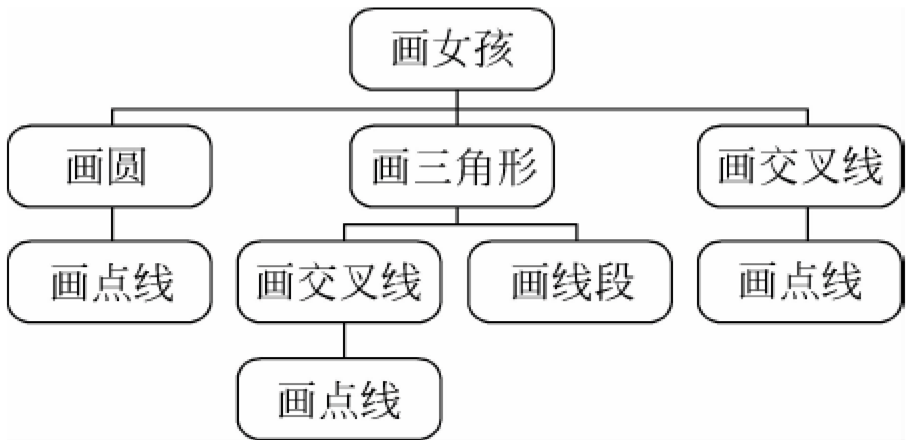


图 6-3-3 画女孩的模块结构图

同样,对画男孩和画房子的过程进行分析,可得到如图 6-3-4 和图 6-3-5 所示的模块结构图。

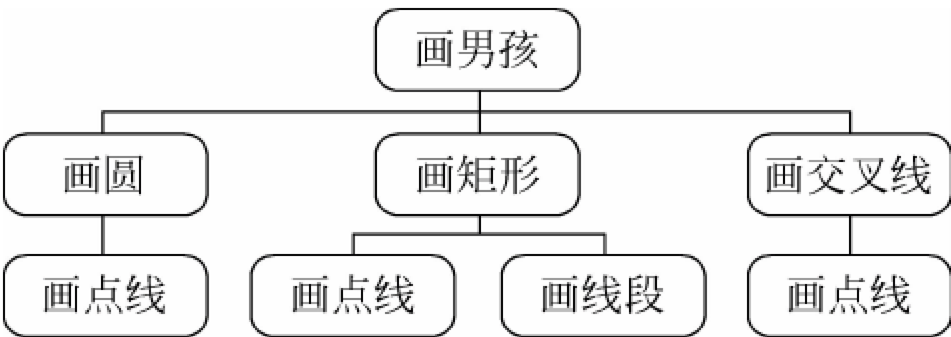


图 6-3-4 画男孩的模块结构图

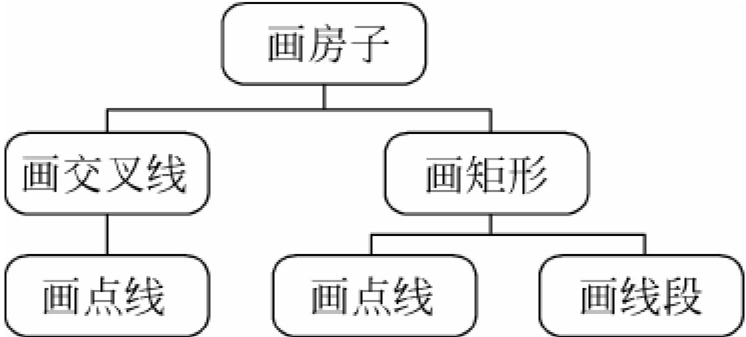


图 6-3-5 画房子的模块结构图

从这三个模块结构图中可以看到,按不同方法反复使用画点线和画线段程序代码可以得到不同的构件(圆、矩形、交叉线、三角形等),这些构件的重复使用又可以得到新的图形。定义好画圆、画矩形、画交叉线,画三角形的构件后,画女孩、画男孩、画房子时就可以减少基本形状实现的重复编码。

2) 设计

① 构造工具函数。

首先构造画点线和连续线段的两个工具函数,在此基础上再来构建其他图形。

- 画点线 drawPoint(start,end):start 和 end 为两个整数,表示点线的两个星号出现的位置。函数的功能是打印一条点线,在 start 和 end 位置上打印两个星号。

具体函数代码如下:

```
def drawPoint(start,end):
    if(start == end):          ##如果 start 和 end 重合,打印一个点
        print(" " * (start - 1) + " * ")
    else:                      ##打印两个点的点线
      print(" " * (start - 1) + " * " + " " * (end - start - 1) + " * ")
```

例如,drawPoint(2,6)将画出一条在位置 2 和 6 上有两个星号的点线:_ * _ _ _ *(其中_表示空格)。

- 画连续线段 drawLine(start,end):函数的功能是从 start 开始到 end 结束,打印一条连续星号构成的线段。

具体函数代码如下:

```
def drawLine(start,end):
    print(" " * (start - 1) + " * " * (end - start + 1))
```

例如,drawLine(2,6)将画一条从位置 2 到位置 6 上有 5 个星号的点线:_ * * * * *(其中_表示空格)。

② 根据工具函数创建圆、矩形、交叉线及三角形构件函数。

```
def drawCilrcle():              ##画圆
    drawPoint(3,5)
    drawPoint(2,6)
    drawPoint(3,5)
def drawInsect():               ##画交叉线
    drawPoint(4,4)
    drawPoint(3,5)
    drawPoint(2,6)
def drawRectangle():            ##画矩形
    drawLine(1,7)
    drawPoint(1,7)
    drawPoint(1,7)
    drawPoint(1,7)
```

```
    drawLine(1,7)
def drawTriangle():              ##画三角形
    drawInsect()
    drawLine(1,7)
```

③ 使用圆、三角形和交叉线构件函数画女孩图形。

```
##画女孩
   drawCilrcle()
   drawTriangle()
   drawInsect()
```

请读者自己完成画男孩和画房子的程序代码。

3）讨论

（1）观察上面画女孩的三条语句，并不能很容易地看出其实现的功能，为提高程序的可读性，还可以定义一个函数 draw_a_girl()，这样程序结构层次更清楚，如图 6-3-6 所示。请思考如何修改程序。

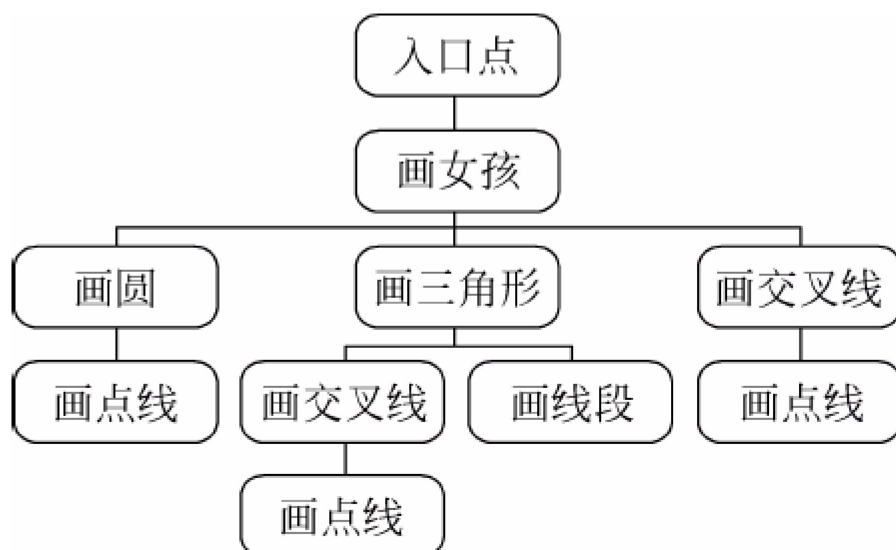


图 6-3-6　改进的画女孩模块结构图

（2）对画男孩和画房子模块同样可做类似的改进。还可以在主程序中根据用户的输入选择画女孩、男孩还是房子。例如，当接收到字符 g(girl)时，调用画女孩模块；接收到字符 b(boy)，调用画男孩模块；接收到字符 h(house)，调用画房子模块。请思考如何修改程序。

（3）进一步思考：能使用以上建立的工具函数和构件函数画出其他图形吗?

2. 自顶向下逐步求精的程序设计

结构化程序设计强调程序设计的规范化和程序结构的模块化，其基本思路是：采用自顶向下、逐步细化的方法把一个复杂问题的求解过程分阶段、分层次进行，每个阶段处理的问题都控制在较易理解和处理的范围内。同时，在程序设计中采用模块化结构，将一个复杂的任务分解为若干相对简单并彼此较独立的模块，还可以将这些模块再细分为若干更小的子模块，以利于“分而治之”、“各个击破”。

程序设计语言中的函数正是实现结构化程序设计必不可少的工具。

【例 6-3-2】 成绩管理系统程序。

程序各模块及关系如图 6-3-7 所示。

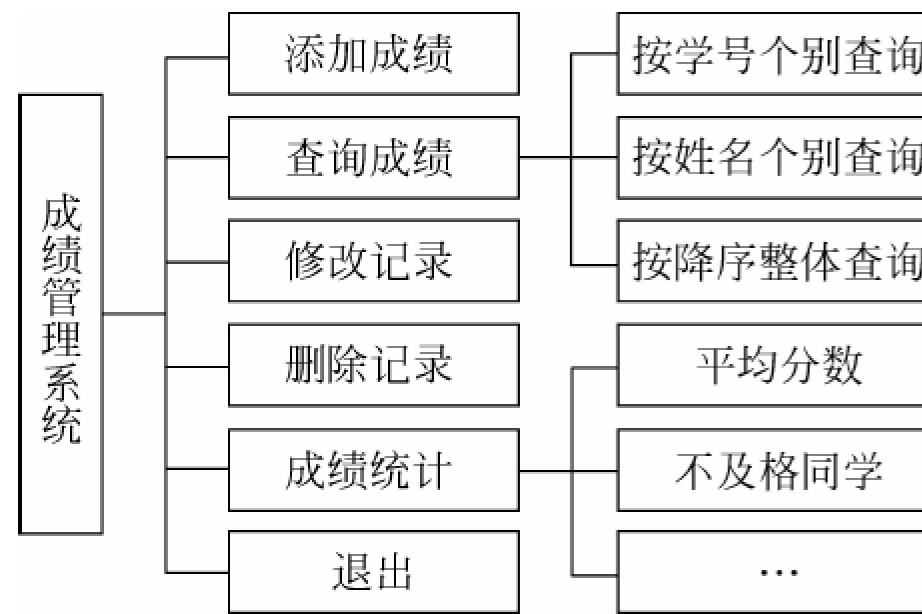


图 6-3-7　成绩管理系统模块

下面仅给出顶部主函数及第一层函数定义框架,按此方法可继续向下扩展,并分别完成各函数模块的具体功能。

```
# function insert score
def insert():
    input("insert() -- -- unfinished." )
# function find score
def find():
    input("find() -- -- unfinished." )
# function edit score
def edit():
    input("edit() -- -- unfinished." )
# function delete score
def delete():
    input("delete() -- -- unfinished." )
# function statistics score
def stat():
    input("stat() -- -- unfinished." )
# function main
def main():
    print(" ************************************ ")
    print(" score management system ")
    print(" ************************************ ")
    print()
    print(" 1.insert score  2.find score")
    print(" 3.edit record   4.delete score")
    print(" 5.statistics    0.quit")
    while True:
        choice = input("please Enter(0 - 5):" )
        if choice == '1':
            insert()
        elif choice == '2':
            find()
        elif choice == '3':
            edit()
        elif choice == '4':
            delete()
        elif choice == '5':
```

```
            stat()
        elif choice == '0':
            break
        else:
            print("Enter error!Choice again.")
    print("Thank you visit!")
#programe entrance
main()
```

说明：

(1) 程序的主控模块为 main()函数，程序入口为最后一条语句(主程序)，这是沿袭了 C 语言的编程风格。因为 C 语言程序都是由函数构成的，不存在主程序，并且规定程序的入口是 main()函数。

(2) 以上各函数的实际功能有待于进一步编程具体实现，这里每个函数中仅给出一条输出信息，便于本程序框架的调试。输出信息并没有使用 print 语句，而是用 input 语句，利用该语句后面所带的提示字符串显示信息，目的是运行时可以使程序显示信息后暂停(等候用户按键后继续)，避免 print 语句执行后信息一晃而过的情况。

程序运行结果如图 6-3-8 所示。

```
Python Shell
File  Edit  Shell  Debug  Options  Windows  Help
*****************************************
        score management system
*****************************************

  1.insert score    2.find score
  3.edit record     4.delete score
  5.statistics      0.quit
please Enter(0-5):1
insert()----unfinished.
please Enter(0-5):2
find()----unfinished.
please Enter(0-5):5
stat()----unfinished.
please Enter(0-5):8
Enter error!Choice again.
please Enter(0-5):0
Thank you visit!
>>>
Ln: 22 Col: 4
```

图 6-3-8　例 6-3-2 运行效果

6.4　模块和 Python 标准库

6.4.1　模块

Python 模块是一个.py 文件，其中包含多个定义的常量和函数代码(以及自定义数据类型、类等)，供其他 Python 程序使用。

我们知道，一个 Python 程序文件的扩展名也是.py，这两者的区别是：程序的设计目的是运行，而模块的设计目的是由其他程序导入并使用。

简单地说,模块就是把常用的一些功能单独放置到一个文件中,方便其他文件来调用。Python 以模块提供的方式,加上它的开源特性,可方便地扩充语言的功能。

1. 内置模块和非内置模块

Python 中的内置函数是通过__builtins__模块提供的,该模块为内置模块,无需手动导入,启动 Python 时系统会自动导入,任何程序都可以直接使用它们。

内置模块中定义了一些软件开发中常用的函数,这些函数实现了数据类型的转换、数据的计算、序列的处理、常用字符串处理等。如第 3 章介绍的 help()函数、round()函数、repr()函数等。

Python 3.3 中的内置函数如表 6-4-1 所示。

表 6-4-1 Python 3.3 的内置函数

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

在 Python 中也可手动导入其他非内置模块,方便地扩充语言功能。

前面第 3 章(3.2.3)中已经介绍了非内置模块导入的几种方式,以及相应方式下函数的引用形式,这里就不再重复介绍了。需要强调的是,使用 from 模块名 import * 语句导入模块,在以后调用函数时可以省略“模块名.”前缀,虽然比较方便,但要注意所引入模块中的函数名等是否与现有系统中的产生冲突。

模块导入后,可以使用内置函数 dir()来查看模块内部的函数名(以及类和常量标识符名称等)。

【例 6-4-1】 使用 dir()函数查看模块信息。

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__']
>>> import math
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__', '__package__', 'math']
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',
```

```
'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
>>>
```

由上例可知，dir()函数若不带参数，将返回当前所有内置模块及已导入的模块名，如果后面带上模块名参数，将返回该模块内的所有函数、常量标识符等名称。

对于内置模块中的函数等，虽不需要手动导入即可使用，也可以用 dir(__builtins__)来查看。

如要进一步具体了解某个函数的作用，可使用 help()函数，例如：

```
>>> help(math.floor)
Help on built - in function floor in module math:
floor(...)
    floor(x)
    Return the floor of x as an int.
    This is the largest integral value <= x.
```

2. 模块的集合——包

在 Python 中，与模块相关的还有一个“包”的概念。包是一个目录，其中包含一组模块和一个_init_. py 文件(记录该目录中的所有. py 文件)。

如果模块存在于包中，使用“import 包名. 模块名”形式导入包中的模块。这时，可使用以下形式访问函数：包名. 模块名. 函数()。

3. 用户自定义模块

除了 Python 系统提供的模块，用户还可以很方便地以文件的形式扩充自己模块库。

【例 6-4-2】 自定义函数库示例。

现以根据三角形三边求面积为例，按以下步骤创建和使用求三角形面积的自定义函数库。

(1) 创建 py 文件，例如 triangle. py，保存在 Python33 目录下，此时 py 文件名就是用户自定义的模块库名。文件的内容为：定义一个 CalArea 函数用于计算三角形面积，函数需要获取三条边的数据，在函数调用时赋值给参变量 a,b,c。

```
import math
def CalArea(a,b,c):
    s = (a + b + c)/2.0
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))
    return area
```

(2) 导入用户自定义的模块库。

```
>>> import triangle
```

(3) 调用用户自定义的函数。

```
>>> area = triangle.CalArea(12,33,25)
>>> area
126.8857754044952
>>>
```

从上面的步骤可以看出,当用户模块文件存放在 python33 的系统目录中时,用户模块的操作与系统非内置模块的操作是一样的。

6.4.2 Python 标准库

随着每个 Python 版本的发布,会同时发布该版本的 Python 标准库。

Python 的标准库十分庞大,其中既有 Python 语言自身特定的类型和声明,包括支持内建数据类型操作的基本模块,如前面提到的用于数学计算操作的 math 模块、为复数提供类似操作的 cmath 模块、实现常用字符串处理的 string 模块以及实现了各种 I/O(输入/输出)形式和内置 open()函数的 io 模块等;也包含很多用于特定领域、帮助用户处理各种工作的模块工具,诸如正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、电子邮件、FTP、XML、GUI(图形用户界面)甚至密码系统等有关操作的模块,这些模块为操作系统、解释器和互联网之间的交互等提供了有效的工具。所有这些模块都得到了充分的测试,可以用来作为应用开发的起点。

以下介绍几个标准库中的基本模块,更多的模块用户可在以后的工作实践中根据需要逐步熟悉掌握。

1. os 模块

os 模块包含了常用的操作系统功能,其常用方法(方法即类中的成员函数)如表 6-4-2 所示。

表 6-4-2 os 库中的常用函数

常用函数	描述
os. name	os. name 方法可获取当前系统平台信息(Windows 下返回'nt',Linux 下返回'posix')
os. linesep	可获取当前平台使用的行终止符(Windows 下返回'/r/n',Linux 使用'/n')
os. getcwd()	获取当前工作目录,即当前 Python 脚本工作的目录路径
os. listdir(path)	返回指定目录下的所有文件和目录名(path 为具体路径)
os. chdir(path)	改变当前路径(path 为要设置的具体路径)
os. makedirs(path)	创建新目录(path 可以是绝对路径或相对路径)
os. removedirs(path)	删除空目录(path 可以是绝对路径或相对路径)
os. system()	os. system()函数用来运行 Shell 命令,可方便调用或执行其他脚本和命令

【例 6-4-3】 os 模块使用示例。

首先使用 import os 导入模块,然后在 Python 提示符下执行 os. system('notepad')命令,可以打开 Windows“记事本”程序。若输入 os. system('notepad NEWS. txt')命令可以打开“记事本”程序并显示当前目录下指定的文本文件(此处为 NEWS. txt)。

如图 6-4-1 所示为 os 模块中部分命令的执行结果。

2. time 和 datetime 模块

Python 提供了多个内置模块用于操作日期时间,如 calendar,time,datetime 等。其中使用 calendar 模块可以将给定年份/月份的日历输出到标准输出设备上。

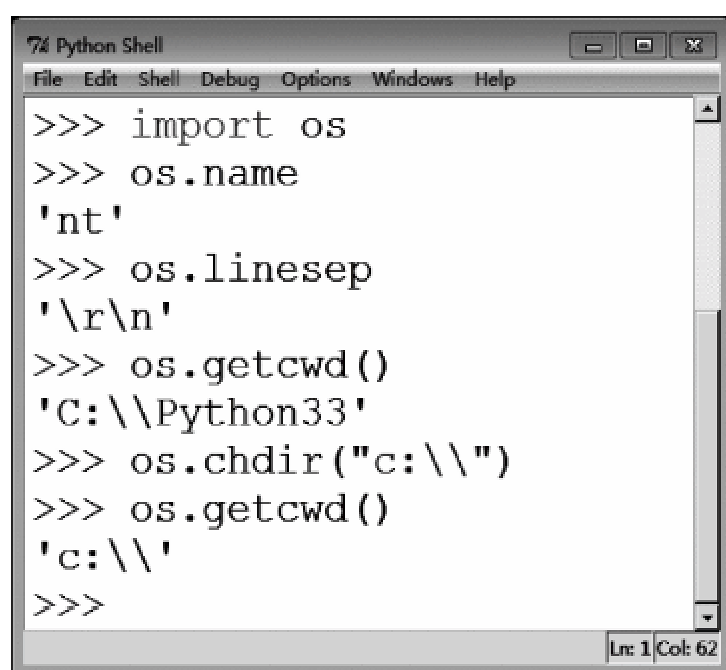


图 6-4-1　os 模块使用示例

【例 6-4-4】　使用 calendar 模块打印月历。

```
>>> import calendar
>>> calendar.prmonth(2014, 8)
    August 2014
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

如要打印年历(例如 2015 年年历),可使用语句:calendar.prcal(2015)。

time 模块提供的接口与 C 语言标准库 time.h 基本一致。相比于 time 模块,datetime 模块的接口则更容易调用些。两者平时使用都较多。下面通过举例简单介绍 time 和 datetime 模块的一些应用。

【例 6-4-5】　time 和 datetime 模块的应用。

① 计算两个日期间隔的天数。

```
>>> import datetime
>>> d1 = datetime.datetime(2014, 2, 16)
>>> d2 = datetime.datetime(2013, 12, 31)
>>> print((d1 - d2).days)
47
>>>
```

② 显示当前日期和时间。

```
>>> datetime.datetime.now()
datetime.datetime(2014, 8, 15, 19, 4, 19, 407150)
>>>
```

说明:

- 这里调用的是 datetime 模块中 datetime 类的方法(即成员函数)now()。两个 datetime 表示的意义不同,不能省略一个。另外,在 datetime 模块中,除 datetime 类外,还有主要用于日期的 date 类和主要用于时间的类 time。

• 在 Python 中也可使用 datetime. datetime. today()显示当前日期和时间,两者效果相同(有兴趣的读者可在 Excel 单元格中分别输入:=today()和=now(),观察在 excel 中这两者的区别)。
• 在 Python 中还可以使用 time 模块中的 localtime()显示当前日期和时间,这将在下面④中讨论。还可使用 time. time()获得当前时间的时间戳(关于时间戳后面会简单提及)。

③ 判断输入的日期是星期几。

```
>>> datetime.datetime(2014,8,15).weekday()
4
>>>
```

④ 格式化日期和时间。

使用 time 模块中的 localtime()函数也可以显示当前日期和时间。

```
>>> import time
>>> time.strftime("%Y-%m-%d %X",time.localtime())
'2014-08-15 18:49:33'
```

在这里使用了 time. strftime()函数对日期和时间格式化。试比较下面不用 time . strftime()的显示结果:

```
>>> time.localtime()
time.struct_time(tm_year=2014, tm_mon=8, tm_mday=15, tm_hour=18, tm_min=50, tm_sec=
29, tm_wday=4, tm_yday=227, tm_isdst=0)
>>>
```

以下介绍 time. strftime()中一些常用的格式参数:

```
%m 表示月份(01—12),d 表示一个月中的第几天(01—31)
%Y(大写 Y)表示 4 位数字的年份,y(小写 y)表示 2 位数字的年份
%X(大写 X)表示时间字符串,x(小写 x)表示日期字符串
```

例如:

```
>>> time.strftime("%y-%m-%d %x",time.localtime())
'14-08-15 08/15/14'
>>>
%H 表示小时(24 小时制,00—23); M 表示分钟数(00—59); S 表示秒
%a 为星期的简写,如星期三为 Web; A 为星期的全写,如星期三为 Wednesday
%b 为月份的简写,如 4 月份为 Apr; B 为月份的全写,如 4 月份为 April
```

例如:

```
>>> time.strftime("%y-%m-%d %A",time.localtime())
'14-08-15 Friday'
```

以上介绍的格式函数 strftime()在 datetime 模块也可使用格式,如下面所示:

```
>>> datetime.datetime.strftime(datetime.datetime.now(), '%Y-%m-%d %H:%M:%S')
'2014-08-15 20:12:02'
>>>
```

⑤ 将字符串转换为时间结构元组。

在 Python 的 time 模块中有以下几种表示时间形式：

a. 时间结构元组(time. struct_time)，时间结构元组共有 9 个元素，依次为年、月、日、时、分、秒、星期(0—6，0 为星期日)、一年中第几天、是否夏令时等(0 为普通，1 为夏令时)。例如前面已举例 time. localtime()得到的就是时间结构元组。

b. 格式化的时间字符串。

c. 时间戳(timestamp)，时间戳是相对于 1970.1.1 00:00:00 以秒计算的偏移量，例如，执行 d=time. time()得到的一个浮点数就是时间戳，可使用 time. ctime(d)转换为字符串。关于时间戳这里不展开讨论。

由④可知，time. strftime()是将日期(时间结构元组)转换为字符串表示，如需将字符串转换为时间结构元组可使用 time. strptime()。

例如：

```
>>> import time
>>> a = "2015 - 10 - 10 23:40:00"
>>> t = time.strptime(a, " %Y- %m- %d %H: %M: %S")
>>> type(t)
<class 'time.struct_time'>
>>> time.strftime(" %y- %m- %d %X",t)
'15 - 10 - 10 23:40:00'
>>> time.strftime(" %y 年 %m 月 %d 日",t)
'15 年 10 月 10 日'
>>>
```

此外，格式参数%c 表示标准的日期格式字符串，例如：

```
>>> t = time.localtime()
>>> type(t)
<class 'time.struct_time'>
>>> d = time.strftime(" %c",time.localtime())
>>> d
'08/15/14 21:31:23'
>>> type(d)
<class 'str'>
>>>
```

⑥ 获取日期时间间隔。

使用 datetime. timedelta()函数可以得到日期/时间的间隔，如显示时间差，或将日期/时间加(减)一个间隔等，例如：

```
>>> a = datetime.datetime(2014,8,15)
>>> c = a - datetime.timedelta(days = 3)
>>> a
datetime.datetime(2014, 8, 15,0,0)
>>> c
datetime.datetime(2014, 8, 12,0,0)
>>>
```

参数除 days 外，还有星期(weeks)以及时(hours)、分(minutes)、秒(seconds)等。例如：datetime.timedelta(hours=5,minutes=8,seconds=10)表示间隔 5 小时 8 分 10 秒。

```
>>> after_a_week = a + datetime.timedelta(weeks = 1,minutes = 8)
>>> after_a_week
datetime.datetime(2014, 8, 22, 0, 8)
>>>
```

3. random 模块

在程序设计中常会遇到需要随机数的情况，Python 的 random 模块提供了产生随机数(以及随机字符)的多种方法。

- random.random()，用于生成一个 0～1 的随机小数。

```
>>> random.random()
0.0794337434342337
>>>
```

- random.randint(a, b)，用于生成一个指定范围内的整数。其中参数 a 是下限，参数 b 是上限。

```
>>> random.randint(1,30)
12
>>> random.randint(1,30)
4
>>>
```

- random.uniform(a, b)，用于生成一个指定范围内的随机浮点数，两参数分别是上限和下限。如果 $a > b$，则生成的随机数 n：$a \leqslant n \leqslant b$；如果 $a < b$，则 $b \leqslant n \leqslant a$。

```
>>> random.uniform(1,30)
8.646522703100882
>>> random.uniform(1,30)
23.62988412891777
>>> random.uniform(30,1)
12.451567610114772
>>>
```

- random.choice(序列对象)，从给出的序列中随机选择一项。序列包括列表、元组和字符串等。

```
>>> random.choice([3,5,7,8,2])
8
>>> random.choice([3,5,7,8,2])
5
>>> random.choice ( ['apple', 'pear', 'peach', 'orange', 'lemon'] )
'lemon'
>>> random.choice ( ['apple', 'pear', 'peach', 'orange', 'lemon'] )
'peach'
>>>
```

- random.randrange([start], stop[, step])，从指定范围(start 到 stop)内，按指定步长(step)递增的集合中获取一个随机数。

```
>>> random.randrange(10, 100, 2)
14
>>> random.randrange(10, 100, 2)
78
>>>
```

说明：

random.randrange(10，100，2)，相当于从[10，12，14，16，…，96，98]序列中获取一个随机数。random.randrange(10，100，2)在结果上与 random.choice(range(10，100，2))等效。

另外，参数 start 和 step 可根据情况省略。

- random.sample(序列，*k*)，从所给序列中随机获取指定长度 *k* 的片断。序列包括元组、列表和字符串等。

```
>>> a = ('a','e','i','o','u')
>>> b = random.sample(a, 3)
>>> c = random.sample(a, 3)
>>> a
('a', 'e', 'i', 'o', 'u')
>>> b
['u', 'a', 'o']
>>> c
['a', 'o', 'i']
```

又如：

```
>>> x = 'hello'
>>> random.sample(x,5)
['e', 'l', 'o', 'h', 'l']
>>> random.sample(x,5)
['h', 'o', 'l', 'l', 'e']
>>>
```

- random.shuffle(x[，random])，用于将一个列表中的元素打乱。

```
>>> p = ["Python", "is", "powerful", "simple", "and so on…"]
>>> random.shuffle(p)
>>> p
['is', 'simple', 'powerful', 'Python', 'and so on…']
>>> random.shuffle(p)
>>> p
['powerful', 'Python', 'is', 'simple', 'and so on…']
```

【例 6-4-6】 模拟洗牌程序。创建一个.py 文件，输入以下程序代码，则每次运行可得不同的数字排列。

```
import random
items = [1, 2, 3, 4, 5, 6]
random.shuffle(items)
print(items)
```

注意：sample()函数不会修改原有序列，但 shuffle()函数将直接改变原有序列。

以上介绍了 Python 标准库中 random 模块产生随机数据的多种方法,除此之外,random 模块还支持三角、β 分布、指数分布、正态分布、γ 分布、高斯分布等非常专业的随机算法,功能十分强大。

【例 6-4-7】 编写一个玩猜数的游戏。由程序产生一个 1~1000 的随机数,玩游戏者可输入最多 10 次猜数。每次如果输入的数不对,可给出偏大偏小提示。如果猜正确,给出恭喜信息,游戏结束;如果 10 次猜数不正确,游戏结束,给出失败信息。

设计要点:

首先建一个函数 echo(),判断所产生的随机数与游戏者所猜数之间的大小关系:猜大返回 1,猜小返回 −1,猜对返回 0。

```
def echo(guess_number,x):
    if x > guess_number:
        return 1
    elif x < guess_number:
        return -1
    else:
        return 0
```

主程序中首先产生一个随机数(当然要导入 random 模块),然后在定义了一个计数变量初值后进入循环。在循环结构中,接收游戏者通过键盘输入的数,然后调用 echo()函数,根据函数返回的结果进行处理:若猜数正确,结束循环,若猜得不对,给出大或小的提示,然后如果次数少于 10 次继续下一轮猜数,如果次数已达 10 次则也结束循环。

循环结束后有两种情况:①已经猜了 10 次且都不正确;②在 10 次内猜对了数。根据这两种情况给出不同的信息。

完整程序代码如下:

```
import random
def echo(guess_number,x):
    if x > guess_number:
        return 1
    elif x < guess_number:
        return -1
    else:
        return 0
gn = random.randint(1,1000)
count = 1
while count <= 10:
    x = int(input("请猜数(第%d次)" % count))
    check = echo(gn,x)
    if check == 0:
        break
    elif check > 0:
        print("猜大了!")
    else:
        print("猜小了!")
    count += 1
if count > 10:
```

```
    print("游戏结束,你失败了.")
else:
    print("恭喜你猜对了,共猜了%d次" % count)
```

以上介绍了 Python 标准库众多模块中的三个模块。Python 标准库是随 Python 附带安装的,熟悉 Python 标准库十分重要,因为如果熟悉这些库中的模块,那么大多数问题都可简单快捷地使用它们来解决。标准库好比一个百宝箱,能为各种常见的任务提供有力的工具和解决方案。

可以在 Python 附带安装文档的"库参考"一节中了解 Python 标准库中所有模块的完整内容,如图 6-4-2 所示。

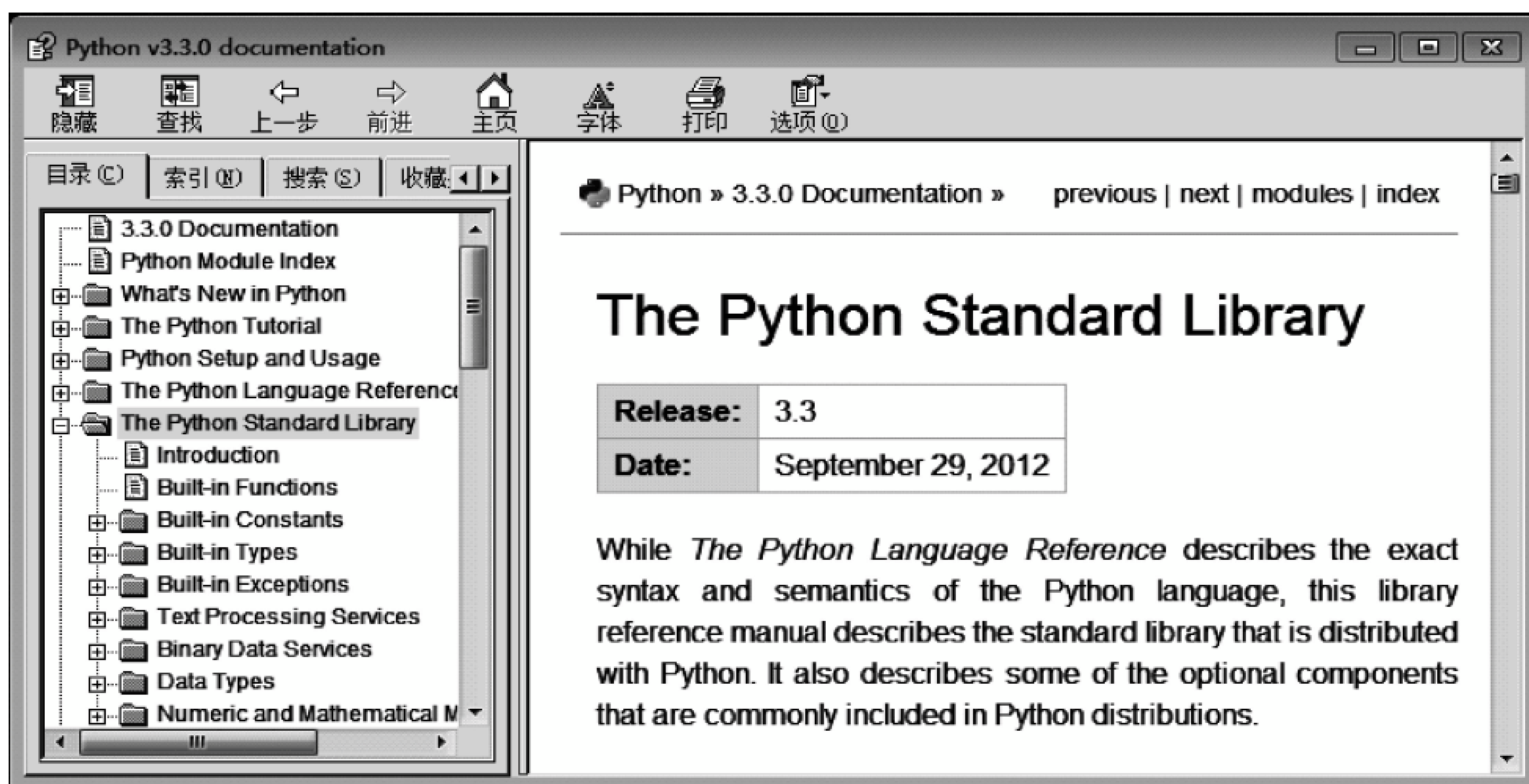


图 6-4-2　使用"帮助"文档了解 Python 标准库中模块信息

当然,作为一个开源软件,除了标准库外,网上还有很多第三方模块包。例如,在科学和工程计算方面,第三方包 NumPy 提供了高效的 n 维数组、基本的线性代数函数和傅里叶变换函数等,SciPy 包提供了用于统计学计算、信号与图像处理、遗传算法等领域运算的函数和工具。这两个包都可以从 www.scipy.org 处获取。此外,如 wxPython——Python 语言的一种优秀的 GUI 图形库、Twisted——网络编程库等,不胜枚举,这也正是 Python 语言吸引人的一大优势。

6.5　本章小结

本章主要介绍程序设计中函数和子程序的概念,以及在 Python 语言中定义和调用函数的方法。本章要点如下:

(1) 函数是程序中完成一定功能的一段独立程序代码,供程序中其他代码调用。使用函数可以使程序的结构清晰,更易于阅读和维护,是结构化程序设计必须具备的重要特征。

(2) 在 Python 语言中,使用 def 关键字定义函数,后接函数名以及圆括号。函数可以带有参数,使应用更灵活。函数定义时的参数在调用时必须给出对应的值,称为实在参数(除非在定义时已经设定了默认参数)。如果参数是列表,其传递到函数中后若列表内容发

生变化将直接影响调用时传递进去的原有列表对象，否则参数是单向传递的，即在函数中对非列表参数进行的操作不会影响函数外面。

(3) 函数中可以通过 return 语句返回值，也可以没有 return 语句或不带返回值。对于有返回值的函数，可以在表达式中使用或作为单独语句调用。如果函数没有返回值，一般只能作为单独语句调用。具有返回值的函数调用也可以作为另一个调用函数的实际参数使用。

(4) 在函数中定义的变量为局部变量，只能作用于本函数，不能用在函数外面。在主程序中定义的变量为全局变量，全局变量既可用在主程序中，也可以在各函数中使用。但程序中过多使用全局变量，将使函数间的耦合变得紧密，破坏函数的独立性。

(5) 大多数程序设计语言除了提供让用户自定义函数的方法和工具外，还提供一些现成的函数让用户直接调用，方便用户使用。Python 以模块库的方法将现成函数等提供给用户。其中随每一个版本都提供了标准库，内有非常丰富的函数资源供用户使用。本章向用户介绍了使用模块库中函数的方法，并通过对标准库中三个模块的具体介绍让读者感受 Python 模块库的强大功能。

此外，函数也是实现递归等算法必不可少的工具。这将在后续章节中进一步讨论。

6.6 习题与思考

6-1 单选题。

(1) 在 Python 中，对于函数定义代码的理解，正确的是：________。

A. 必须存在形参

B. 必须存在 return 语句

C. 形参和 return 语句都是可有可无的

D. 形参和 return 语句要么都存在，要么都不存在

(2) 在 Python 中，对于函数中 return 语句的理解，错误的是：________。

A. 一定要有 return 语句

B. 可以有多条 return 语句，但只执行一条

C. return 可以带返回参数

D. return 可以不带返回参数

(3) 函数一般不能________。

A. 嵌套调用　　B. 嵌套定义

C. 自己调用自己　　D. 没有返回值

(4) 在一个函数中若局部变量与全局变量同名，则________。

A. 局部变量屏蔽全局变量

B. 全局变量屏蔽局部变量

C. 该两个局部变量和全局变量都不可使用

D. 该两个局部变量和全局变量在函数中互不干扰，各自发挥作用

(5) 假设 Sport 模块中有 Run 函数，则在执行了语句：from Sprot import Run 后，要调用 Sport 中的 Run 函数，应该使用________。

A. Sport(Run)　　B. Sport. Run()

C. Sport()　　　　　　　　D. Run()

6-2　形式参数和实际参数有什么区别?

6-3　局部变量和全局变量有什么区别?

6-4　在 Python 函数定义中,是否允许只有函数定义头部,后面不带任何语句?

6-5　在 Python 程序中,是否允许函数 A 调用函数 B,函数 B 再调用函数 A?

6-6　编一个程序,输出如图 6-6-1 所示的图形。要求使用函数打印每行"+",每行打印数量由函数参数传递。主程序中通过循环结构控制打印行数。

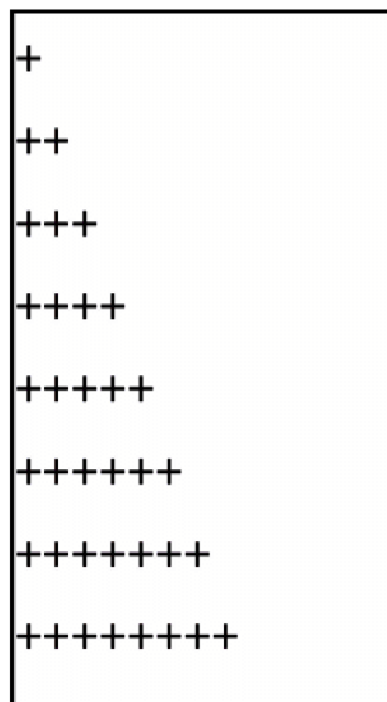


图 6-6-1　习题 6-6 输出

6-7　编一个求平方根函数,允许使用 math 模块的相应函数,但必须进行判断:如果所给数为负数,则显示无实数解信息。在函数外接收键盘输入的数并显示其平方根。

6.7　实验　函数和模块的使用

6.7.1　实验目标

(1) 掌握在 Python 中自定义函数及其调用的方法。

(2) 了解参数传递和程序中变量的作用范围。

(3) 熟悉 Python 基本模块中常用函数的使用。

6.7.2　实验范例

1. 创建和调用函数

(1) 在 Python 的 IDLE 下直接输入以下代码创建函数:

```
>>> def star(m,n):
    for i in range(m):
        print('*'*n)
```

然后分别用以下语句调用该函数:

```
>>> star(3,2)
>>> star(5,6)
>>> star(4,20)
```

(2) 输入以下代码创建函数:

```
>>> def paint(m,s):
    print(s*m)
```

然后分别用以下语句调用该函数:

```
>>> paint(3,'*')
>>> paint(8,'%+')
```

(3) 输入以下代码创建函数:

```
>>> def check(a):
    if a > 0:
        print("> 0")
    elif a < 0:
        print("< 0")
    else:
        print(" == 0")
```

然后分别用以下语句调用该函数:

```
>>> check(5)
>>> check(-2)
>>> check(0)
```

(4) 在 Python 的 IDLE 下直接输入以下代码创建函数:

```
>>> def avg(a,b):
    return (a + b)/2
```

然后分别用以下语句调用该函数:

```
>>> print(avg(4,6))
>>> x = avg(3,6)
>>> x   # 显示 x 的值
>>> y = avg(3,avg(5,7))
>>> y   # 显示 y 的值
```

(5) 按以下方式修改上题函数:

```
>>> def avg(a,b):
    return (a + b)/2
    return (a + b)
```

再用以下语句调用该函数,观察结果是否改变:

```
>>> print(avg(4,6))
>>> x = avg(3,6)
>>> x   # 显示 x 的值
```

结论:可以有多条 return 语句,但只执行一条。

(6) 按以下方式修改 avg()函数:

```
>>> def avg(a,b = 0):
    return (a + b)/2
```

再用以下语句调用该函数,观察结果:

```
>>> avg(6,7)
>>> avg(6)
```

结论:函数可以带默认参数。

(7) 按以下方式创建函数：

```
>>> def more(a,b):
    return a + b, a - b
```

再用以下语句调用该函数，观察结果：

```
>>> more(2,3)
>>> x,y = more(7,3)
>>> x,y
```

结论：return 语句可以返回多个值。

(8) 按以下方式修改上题函数：

```
>>> def more(a,b):
    return
```

再用以下语句调用该函数，观察结果：

```
>>> more(2,3)
>>> z = more(7,3)
>>> z
```

结论：return 语句后面可以没有返回值，此时函数的值为 None。

2. 使用标准库中的函数

1) 使用 math 模块的数学函数

在 Python IDLE 下直接输入：import math。

然后输入以下表达式理解 math 中函数的使用：

```
math.sqrt(2 * 2 + 3 * 3)、math.log10(100)、math.exp(2)、math.e
math.pow(2.5,2)、math.sin(math.pi/2)、math.tan(math.pi/4)、
math.floor(2.5)、math.ceil(2.5)、math.fmod(4,3)、math.fabs(-23.56)
```

2) 使用 calendar 模块打印年历

在 Python IDLE 下直接输入：import calendar。

然后输入 calendar.prcal(2015)，显示 2015 年年历。

3) 使用 random 模块生成随机数函数

- 在 Python IDLE 下直接输入：import random。

然后输入三次 random.random()，观察每次的值。

- 输入三次 random.randint(1,10)，观察每次的值。
- 输入：result1 = [random.randint(1,100) for i in range(10)]
  result2 = [random.randint(1,100) for i in range(10)]
  result3 = [random.randint(1,100) for i in range(10)]

然后分别显示 result1、result2 和 result3。

- 输入：for i in range(4):
  print(random.sample([1,2,3,4,5,6,7,8],8))，观察结果。
- 输入三次(可用复制/粘贴方法)：print(random.sample(['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday', 'Saturday', 'Sunday'], 7))，观察结果。

4) 使用 sys 模块中的函数

在 Python IDLE 下直接输入：import sys。

- 然后输入：print('当前的 python 目录是：'+sys.prefix) #了解当前路径
- 输入：sys.path.append('c:\\exercise') #添加路径
- 输入：print(sys.path) #显示全部路径

上述操作结果如图 6-7-1 所示。

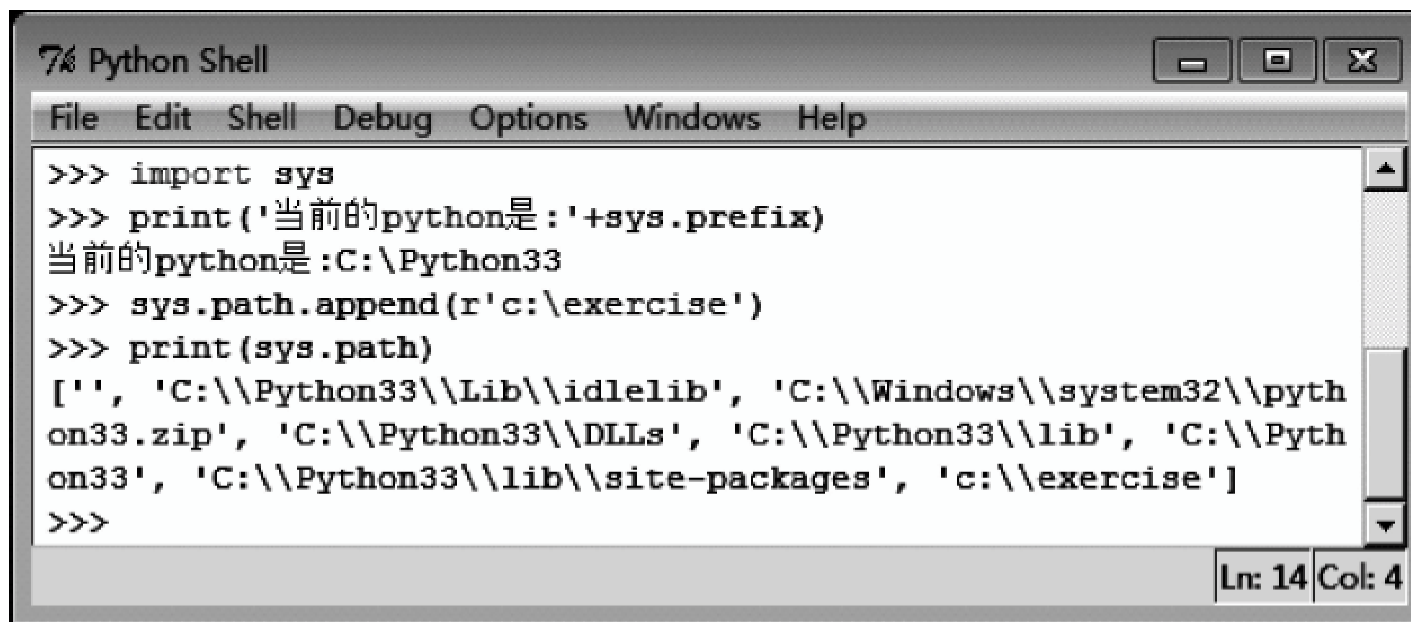


图 6-7-1 使用 sys 模块

3. 以下两程序的功能为求输入两数之差的绝对值，请改正程序中的错误。

```
def absolute(a,b):
    if a>b:
        return a-b
    else:
        return b-a
x1 = input("a = ")
x2 = input("b = ")
print("|a-b| = ",absolute)
```

```
def absolute(a,b):
    if a>b:
        c = a-b
    else:
        c = b-a
    return
x1 = int(input("a = "))
x2 = int(input("b = "))
print("|a-b| = ",absolute(a,b))
```

分析：

左边程序①缺函数调用参数；②注意形参和实参数据类型匹配。

所以最后一句应改为 print("|a-b|=",absolute(int(x1),int(x2)))。

右边程序①函数调用参数不对，应改为 print("|a-b|=",absolute(x1,x2))。

② 函数无返回值，所以 return 语句应改为 return c。

4. 使用 os 模块。首先用“资源管理器”在 C 盘下创建一个新文件夹 mydir，然后在 Python 的 IDLE 界面下按以下要求输入语句并查看结果。

(1) 导入 os 模块。

(2) 使用 os 模块查看当前路径。

(3) 使用 os 模块在 mydir 下面新建一个文件夹 d1。

(4) 使用 os 模块将当前路径改为 C:\mydir。

(5) 使用 os 模块在当前路径下新建一个文件夹 d2。

(6) 使用 os 模块查看当前路径下的所有文件和目录名。

(7) 使用 os 模块将当前路径改为 C:\。

(8) 使用 os 模块删除 mydir 文件夹,并通过“资源管理器”检查操作是否完成。

(9) 使用 os 模块删除 d1、d2 文件夹,并通过“资源管理器”检查操作是否完成,并再次检查 C:\ mydir 文件夹是否存在。

```
>>> import os
>>> os.getcwd()
'C:\\Python33'
>>> os.makedirs("c:\\mydir\\d1")
>>> os.chdir("c:\\mydir")
>>> os.getcwd()
'c:\\mydir'
>>> os.makedirs("d2")
>>> os.listdir()
['d1', 'd2']
>>> os.chdir("c:\\")
>>> os.removedirs("c:\\mydir")
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    os.removedirs("c:\\mydir")
  File "C:\Python33\lib\os.py", line 295, in removedirs
    rmdir(name)
OSError: [WinError 145] 目录不是空的.: 'c:\\mydir'
>>> os.removedirs("c:\\mydir\\d1")
>>> os.removedirs("c:\\mydir\\d2")
>>> os.removedirs("c:\\mydir")
```

5. 编写一个程序,首先定义一个接收字符串参数的函数,其功能为统计传过来的字符串中的字母、数字和其他字符的个数。在函数外输入字符串并输出统计结果。

程序代码:

```
def count(str):
    alph = digit = others = 0
    for i in range(0,len(str)):
        if str[i]>= 'a' and str[i]<= 'z' or str[i]>= 'A' and str[i]<= 'Z':
            alph += 1           #统计字母个数
        elif str[i]>= '0' and str[i]<= '9':
            digit += 1          #统计数字个数
        else:
            others += 1         #统计其他字符个数
    return alph,digit,others
x = input("please input strings:")
a,d,o = count(x)                #函数返回三个值,分别依序赋给 a,d,o 三个变量
print("string:",x)
print("letters:",a,",digits:",d,",others:",o)
```

分析:从本例可以看出函数可以返回多个参数。函数中 return 语句的参数之间用逗号分隔。

6. 编写一个程序,利用公式 $e=1+1/1!+1/2!+1/3!+\cdots+1/n!$ 求自然对数 e 的近似值,其中求阶乘要使用函数,n 值在运行时由键盘输入。

程序代码：

```
def fact(n):                    #定义求 n!的函数
    factorial = 1
    for counter in range(1,n+1):
        factorial *= counter
    return factorial
s=1
n=int(input("input n:"))
for i in range(1,n+1):
    s+=1/fact(i)
print("e=1+1/1!+1/2!+…+1/n!=",s)
```

7. 编写一个程序，求 $ax^2+bx+c=0$ 的实数根(x_1、x_2)，a、b、c 由键盘输入。

要求建两个函数：delta()函数用于判断 b^2-4ac 是否大于等于 0，该函数返回逻辑值(当 b^2-4ac 小于 0 时返回 False，否则返回 True)。real_root()函数用于求 b^2-4ac 大于等于 0 时的实根 x_1、x_2(x_1、x_2 为全局变量)，该函数无需返回值。

主程序按顺序完成以下操作：接收输入的系数 a、b、c，调用 delta()函数判断，如果函数返回值为 True，调用 real_root()函数求实根，最后输出 x_1、x_2 的值(保留两位小数)。

程序运行结果参考如下：

```
a=2
b=-9
c=9
x1=3.00,x2=1.50
>>> ================================ RESTART ============================
a=1
b=2
c=1
x1=-1.00,x2=-1.00
>>>
```

程序代码：

```
from math import *
x1=x2=dt=0
def delta(a,b,c):
    global dt
    dt=b*b-4*a*c
    if dt>=0:
        return True
    else:
        return False
def real_root(a,b,c):
    global x1,x2
    if dt>0:
        x1=(-b+sqrt(dt))/(2*a)
        x2=(-b-sqrt(dt))/(2*a)
    else:
        x1=x2=(-b)/(2*a)
a=int(input("a="))
```

```
b = int(input("b = "))
c = int(input("c = "))
if delta(a,b,c):
    real_root(a,b,c)
    print("x1 = %.2f,x2 = %.2f" % (x1,x2))
else:
    print("no real root!")
```

分析：本例展示了全局变量的使用。

6.7.3 实验内容

（1）以下两程序的功能均为：求输入两数的平均值，请分别改正程序中的错误（要求主程序中的两条赋值语句不可修改）。

```
def aver(a,b):
    d = (a + b)/2
    return
a = int(input("a = "))
b = int(input("b = "))
print("average = ",aver)
```

```
def aver(int(a),int(b)):
    d = (a + b)/2
x1 = input("a = ")
x2 = input("b = ")
print("average = ",d)
```

（2）对上题修改好的求平均值程序进行改进，函数改名为 funct，并增加一个参数 c：如果 c 等于 0 返回平均值，c 等于 1 返回两数之和，c 为其他值提示“输入出错!”。

（3）分析和运行本章例 6-2-5 求阶乘程序。在此基础上改进程序，求 1—5 的阶乘之和。即 1!+2!+3!+4!+5!=?

（4）老王卖西瓜，每天只卖总数的一半多两个。已编一程序：输入西瓜总数（小于 2000 个），输出所需卖的天数。根据以上要求及图 6-7-2 输出。在下面程序的空项中填入合适的内容并调试通过程序。

```
def watermelon(________):
    day = 0
    x1 = int(d)
    while x1 > 1:
        x2 = x1 - (int(x1/2) + 2)
        x1 = x2
        day += 1

    ________
x = input("Enter total number:")
while ________(x) in (range(2000)):
    print("days:",________(x))      #call function
    x = input("Enter total number:")
```

【提示：本程序在函数调用时，通过 x 向函数传递西瓜的总数，函数执行后返回所需天数供打印。程序可反复计算，直到输入西瓜总数不在指定范围，运行结果如图 6-7-2 所示。】

（5）在 Python 3.3 环境下实现例 6-3-1 绘图程序。

① 输入工具函数代码。

② 输入构件函数代码。

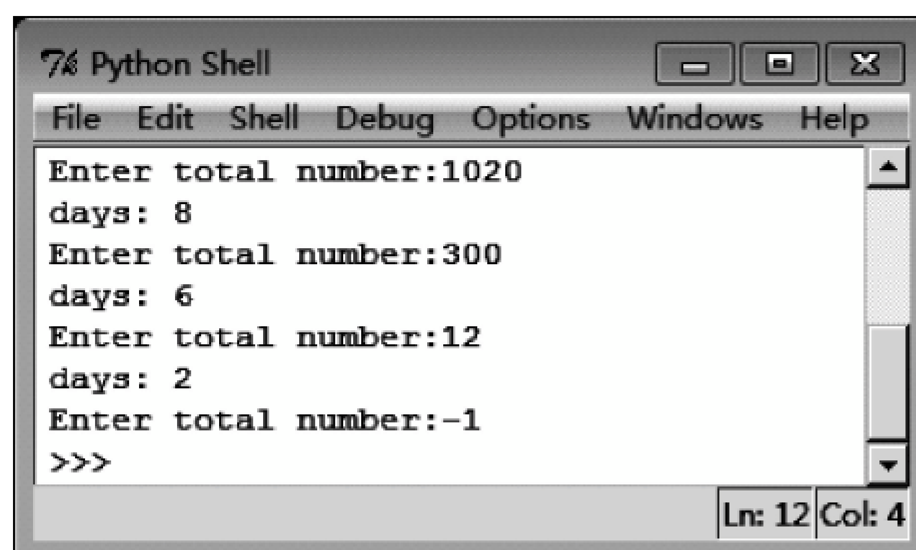


图 6-7-2　第 5 题程序运行效果

③ 在主程序中调用以上相关函数画出女孩、男孩和房子。

④ 尝试使用以上相关函数画出其他图形。

(6) 有人用以下语句计算明天的日期：datetime. datetime. today()＋1(假设已经导入 datetime 模块)，系统提示出错。请用 datetime. timedelta()函数加以修改。

第7章 算法分析与设计

算法是描述程序行为的语言，是一种应用于计算机科学领域，已被实践者广泛接受的理论，是一种让程序最为简洁的思考方式。因此，利用计算机求解实际问题离不开算法研究，就像离不开程序一样。一般而言，算法分析与设计是关于计算机程序性能的研究，它主要包含两个方面的内容：一是如何设计算法，二是如何分析算法的有效性并研究算法是否可行。如前所述，算法设计的任务是对各类具体问题设计良好的算法及研究算法设计的规律和方法。常用的算法有：枚举法、递归法、回溯法、贪心法、分治法等。

7.1 算法性能分析

算法性能分析的主要目标是确定算法的性能特征，对一个算法占用的计算时间和存储空间做定量分析，进而探讨某种算法对求解的问题是否适用以及如何改进给出的算法和程序。

7.1.1 性能分析的重要性

无论是设计算法还是选择算法，算法分析都是重要的：一是可用于比较各种算法的优劣，二是能够准确地确定算法是否可行。算法分析包括时间性能分析和空间性能分析两部分。

1. 从计算时间做定量分析的重要性

- 确保算法的可行性。
- 根据时间需求，从多种不同的解决方案中选取适合的算法。
- 根据运行时间上限，确定某种算法是否为用户可接受的。

2. 从存储空间做定量分析的重要性

- 确定计算机系统是否有足够的内存运行。
- 估计解决问题的最大规模。

空间性能分析的基本概念和方法与时间性能分析是相似的，由于计算机内存容量的不断扩大，从一般意义来说空间上越来越能满足问题的需求。因此，这里的分析主要强调时间性能。

7.1.2 举例说明算法的时间性能分析与量度

估计一个算法的时间性能(也称时间复杂性)，首先需要确定该算法包含了哪些基本运算以及执行这些运算所用的时间。通常算术运算及比较、赋值等操作被看作基本操作，并约

定它们所使用的时间都是一个单位。因此,可以用算法所需基本操作的执行次数来量度算法的时间效率,这种方法独立于特定的计算机系统,与所用的机器、程序设计语言无关,完全由算法本身确定。

为了更容易理解算法所表达的含义,以下所有实例均采用伪代码的形式来描述。

【例 7-1-1】 实数、向量和矩阵的加法。

(1) 实数相加算法。

伪代码:

```
Input: x,y
Output: z
begin
z←x + y                         //一次加法,一次赋值
end                             //两次基本操作
```

(2) 向量相加算法。

伪代码:

```
Input: x,y
Output: z
begin
for i←1 to n do                 //n 循环(n 次赋值)
  z[i]←x[i] + y[i]              //一次加法,一次赋值
  repeat
end                             //3n 次基本操作
```

(3) 矩阵相加算法。

伪代码:

```
Input: x,y
Output: z
begin
for i←1 to n do                 //n 循环(n 次赋值)
  for j←1 to n do               //n 循环(n² 次赋值)
    z[i,j]←x[i,j] + y[i,j]      //一次加法,一次赋值
  repeat
repeat
end                             //n + 3n² 次基本运算
```

分析:这里的关键操作是加法和赋值。在向量相加运算中,for 语句的每一遍循环都执行一次加法运算,两次赋值操作,所以总运算次数为 $2n$。在矩阵相加运算中,第二个 for 循环执行 n 次,加法运算执行 n^2 次,赋值操作执行 $n+2n^2$ 次。

7.1.3 计算时间的渐近估计表示

1. 时间性能的分析方法

确定算法执行次数是为了比较不同算法的时间性能。一般而言,任何算法,输入规模越大,需要的运行时间越长。但对于小规模的输入不同算法在运行时间上几乎没有差别,不足以将高效的算法和低效的算法区分开来。因此,对算法时间性能的评估通常是针对充分大

的输入规模来进行的。

设 $T(n)$是输入规模为 n 的某一算法的时间性能函数。一般来说，当 n 单调增加趋于∞时，$T(n)$也将单调增加趋于∞。如果存在函数 $T'(n)$，使得当 $n\to\infty$时有$(T(n)-T'(n))/T(n)\to 0$，则称 $T'(n)$是 $T(n)$当 $n\to\infty$时的渐近性态。因为在数学上，$T'(n)$是 $T(n)$当 $n\to\infty$时的渐近表达式，$T'(n)$可以是 $T(n)$中略去低阶项所留下的主项，所以它无疑比 $T(n)$来得简单。

这样，就给出了分析算法性能的简约方法，即只考虑当问题的规模充分大时，算法性能在渐近意义下的阶。为此引入常用的渐近符号 O。

【O 定义】 如果存在两个正常数 c 和 n_0，对于所有的 $n\geqslant n_0$，有

$$|f(n)|\leqslant c|g(n)|$$

则记作 $f(n)=O(g(n))$。

因此，当称一个算法具有 $O(g(n))$的计算时间时，指的就是如果此算法用 n 值不变的同一类数据在某台机器上运行时，所用的时间总是小于$|g(n)|$的一个常数倍。

$g(n)$是计算时间 $f(n)$的一个上界函数，$f(n)$的数量级就是 $g(n)$。

【定理】 若 $A(n)=a_m n^m+\cdots+a_1 n+a_0$ 是一个 m 次多项式，则 $A(n)=O(n^m)$。

证明：取 $n_0=1$，当 $n\geqslant n_0$ 时利用 $A(n)$的定义和一个简单的不等式，有

$$\begin{aligned}|A(n)| &\leqslant |a_m|n^m+\cdots+|a_1|n+|a_0|\\ &\leqslant (|a_m|+|a_{m-1}|/n+\cdots+|a_0|/n^m)n^m\\ &\leqslant (|a_m|+|a_{m-1}|+\cdots+|a_0|)n^m\end{aligned}$$

取 $c=|a_m|+|a_{m-1}|+\cdots+|a_0|$，定理得证。

定理表明，变量 n 的固定阶数为 m 的任一多项式，与此多项式的最高阶 n^m同阶。因此，一个计算时间为 m 阶多项式的算法，其时间都可以用 $O(n^m)$来表示。

例如，一个算法的数量级为 $c_1 n^{m1}, c_2 n^{m2}, \cdots, c_k n^{mk}$ 的 k 个语句，算法的数量级及计算时间就是 $c_1 n^{m1}+c_2 n^{m2}+\cdots+c_k n^{mk}=O(n^m)$。其中，$m=\max\{m_i|1\leqslant i\leqslant k\}$。

2. 算法的分类

从计算时间上考虑，算法可分为两类：

(1) 多项式时间算法(Polynomial Time Algorithm)：可用多项式来对其计算时间界限的算法。以下几种计算时间的多项式是最为常见的，分别按占用的时间由小到大排列：

$O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)$

(2) 指数时间算法(Exponential Time Algorithm)：计算时间用指数函数界限的算法。

$$O(2^n)<O(n!)<O(n^n)$$

【例 7-1-2】 棋盘上的麦粒问题。

印度有一个传说：舍罕王打算奖赏国际象棋的发明人——宰相西萨·班·达依尔。国王问他想要什么，他对国王说："陛下，请您在这张棋盘的第一个小格里，赏给我一粒麦子，在第二个小格里给两粒，第三小格给四粒，以后每一小格都比前一小格加一倍。请把摆满棋盘上 64 格的麦粒，赏给我吧！"

国王就命令给他这些麦粒，当士兵把麦子搬来开始计数时，国王才发现：即使把粮仓的麦粒全拿来，也摆不满 64 个棋盘格。那么，宰相要求得到的麦粒总数是多少？

麦粒总数为 $1+2^1+2^2+2^3+\cdots+2^{63}=2^{64}-1=18\,446\,744\,073\,709\,551\,615$。

从这个例子可以看出 2^n 指数级的增长是十分巨大的。

表 7-1-1 列出了各种时间性能函数算法在机器上运行时所花费的时间。其中,n 为问题输入长度,取不同的值用于比较各种时间性能函数在输入规模增大时的表现。

表 7-1-1　对于算法分析具有重要意义的函数值

n	$\log_2 n$	$n\log_2 n$	n^2	n^3	2^n
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65 536
32	5	160	1024	32768	4 294 967 296

可以看出,一个需要指数级操作次数的算法只能用来解决规模非常小的问题。当输入规模逐渐增大时,指数时间性能函数(如 2^n)的运行时间呈爆发式增长。因此,多项式时间算法在时间性能上一定优于指数时间算法。

3. 算法效率的分析方式

对算法进行效率分析的方式主要有以下三种。其中最坏情况分析是最重要的,因为它涵盖了同一规模的所有可能输入情况,算法的运行时间不会比最坏情况下更长。

(1) 最坏情况分析:是指在输入规模为 n 时,算法在最坏情况下的效率,如最长运行时间。

(2) 最好情况分析:是指在输入规模为 n 时,算法在最优情况下的效率,即最优特例的运行时间,它是无法说明算法在一般意义下的真实效率的。

(3) 平均情况分析:是指在“典型”或“随机”输入的情况下,算法具有的效率。

7.2　查 找 问 题

查找和排序是两种比较重要的问题类型。查找又称为搜索,是指从大量已存储信息中检索某条或多条信息的一项基本运算,它是很多计算任务的本质所在。查找的应用很广泛,它涉及各种各样的运算,如搜索引擎可用于查找网络中包含指定关键字的文档。

7.2.1　查找最大数最小数

1. 问题的提出

查找最大最小数是日常生活中经常会碰到的问题,例如统计班级和年级学生的最高最低分。对于这类问题可把它们简化为从 n 个元素中查找最大和最小元素。

2. 问题的解决思路

求最大数和最小数的解题方法是类似的。首先需要确定 n 个输入元素的存储结构,对于批量数据一般是通过数组这类结构(如 Python 的列表)来存放的;然后再按照一定的次序从数组的第一个元素(或最后一个元素)开始逐一与后面(或前面)的元素比较,这可以通

过循环结构来实现。需要特别注意的是最大或最小值应保存到对应的变量中，以便于比较和最后的输出。

3. 解决问题过程的描述

```
算法 MaxMinElement
//求给定数组中的最大最小元素
//输入：实数数组 A[0:n－1]
//输出：A 中的最大元素 maxval 和最小元素 minval
maxval ←A[0]
minval ←A[0]
for i ← 1 to n－1 do
    if A[i] > maxval
        maxval ← A[i]
    if A[i] < minval
        minval ← A[i]
```

4. 算法性能的考虑

- 基本操作为比较和赋值。
- 输入规模是数组长度 n。
- 在最好、最坏情况下，元素的比较次数都是 $2(n-1)$。

算法的效率为 $O(n)$。

总而言之，分析算法效率的通用方案为：

- 决定用哪些参数作为输入规模的量度。
- 找出算法的基本操作。
- 检查基本操作的执行次数是否只依赖于输入规模。
- 建立一个算法基本操作执行次数的求和表达式。
- 计算时间的渐近估计表示。

7.2.2 查找特定数

1. 问题的提出

查找是指从一组记录集合中找出满足给定条件的记录。如在电话号码簿中查找某人的联系方式，在图书馆查找某本书籍等。在讨论查找时，通常假设被查找的对象是由一组同一类型的记录构成的集合，称这个集合为查找表。关键字是指记录中的某个数据项，用它可以标识一个记录。若此关键字可以唯一地标识一个记录，则称此关键字为主关键字；反之，把可以识别若干记录的关键字称为次关键字。因此，具体地说，查找是根据某个给定的值，在查找表中查找一个其关键字值等于给定值的记录。若表中存在这样的一个记录，则称查找成功；若表中不存在关键字值等于给定值的记录，则称查找失败。

查找技术可分为静态查找表技术、动态查找表技术和哈希表技术。我们学习的重点是解决问题的思维方法，因此这里以静态查找表中顺序查找和折半查找为例进行介绍。为简便起见，假设查找表中的记录为实数类型，记录的关键字即为该实数，记录的个数为 n，存放在数组 A 中。

2. 顺序查找技术的解决思路

从查找表的一端开始,逐个将记录的关键字值和给定值进行比较,如果某个记录的关键字值和给定值相等,则称查找成功;否则,说明查找表中不存在关键字值为给定值的记录,则称查找失败。

3. 解决问题过程的描述

```
顺序搜索算法 SequentialSearch。
//在数组 A[0:n-1]中搜索给定值 x,若找到则返回所在的位置,否则返回-1
//输入:实数数组 A[0:n-1],给定值 x
//输出:数组的下标 i 或-1
i ← 0
while i < n and a[i]≠x do
    i ← i+1
if i<n return i
else return -1
```

4. 算法性能的考虑

- 基本操作为比较(特别是搜索值的比较)和赋值。
- 输入规模是数组长度 n。
- 算法的效率为 $O(n)$。

在顺序查找中,关键字的比较次数取决于所查数据(记录)在数组(表)中的位置。如查找记录 $A[0]$时,仅需比较一次,而查找记录 $A[n-1]$时,则需比较 n 次。若关键字不在表中,则必须经过 n 次比较后才能确定查找失败。这个结果表明顺序查找的查找长度是与记录的个数 n 成正比的。因此,顺序查找的优点是算法简单,且对表的结构没有任何要求。它的缺点是查找效率低,当表中元素个数比较多时,不宜采用顺序查找。

5. 折半查找技术的解决思路

折半查找是效率很高的查找方法,但被查找的数据必须是有序的。首先将待查的 x 值与有序表 $A[0]\sim A[n-1]$的中间位置 mid 上的结点的关键字进行比较,若相等,则查找完成;否则,若 $A[\text{mid}]>x$,则说明待查找的结点只可能在左表 $A[0]\sim A[\text{mid}-1]$,只需在左子表中继续查找;若 $A[\text{mid}]\leqslant x$,则在右子表 $A[\text{mid}+1]\sim A[n-1]$继续查找。这样,经过一次关键字的比较就缩小了一半的查找区间。继续按上述方法进行查找,直到找到关键字为 x 的元素或当前查找区间为空(即表明查找失败)为止。

【例 7-2-1】 设有一个 13 个记录的有序表的关键字值如下:

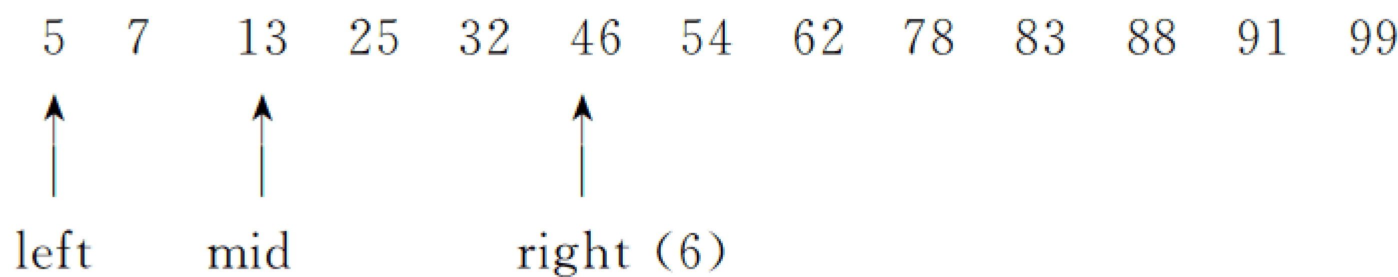
5 7 13 25 32 46 54 62 78 83 88 91 99

假设指针 left 和 right 分别指示待查元素所在区间的下界和上界,指针 mid 指示区间的中间位置。

查找关键字值为 32 的过程如下:

```
5   7   13   25   32   46   54   62   78   83   88   91   99
↑                           ↑                               ↑
left(1)                     mid(7)                          right(13)
```

取 mid 位置的关键字值 54 与 32 作比较，显然 32<54，故要查找的 32 应该在前半部分，所以下次的查找区间应变为[1,6]，即 left 值不变仍为 1，right 的值变为 mid－1＝6，求得 mid＝3(整除)。



取 mid 指示位置的关键字值 13 与给定值 32 作比较，显然 13<32，故要查找的 32 应该在后半部分，所以下次的查找区间应变为[4,6]，即 left 值变为 mid＋1＝4，right 值不变仍为 6，求得 mid＝5。

取 mid 指示位置的关键字值 32 与给定值 32 作比较，显然是相等的，说明查找成功。所查元素在查找表中的位置即为 mid 所指示的值。

6. 解决问题过程的描述

```
折半搜索算法 BinarySearch(这里假定被查找的数组已经是单调递增的)。
//在 A[0]<=A[1]<=…<=A[n-1]中搜索给定值 x,若找到则返回所在的位置,否则返回-1
//输入: 已排好序的实数数组 A[0:n-1],给定值 x
//输出: 数组的下标或-1
left ← 0
right ← n-1
while left <= right do
    middle←(left+right)/2        //整除
    if(x==A[middle]) return middle
    if(x > A[middle]) left=middle+1
    else right=middle-1
return -1
```

7. 算法性能的考虑

while 的每次循环(最后一次除外)都将以减半的比例缩小搜索范围，所以，该循环在最坏的情况下需要执行 $O(\log n)$次。可以通过以下描述方法来理解性能函数为何为 $O(\log n)$ 即 $O(\log_2 n)$。

折半查找的过程可以用判定树来描述，对于上面记录个数为 13 的实例，先画出判定树，如图 7-2-1 所示。

下标	0	1	2	3	4	5	6	7	8	9	10	11	12
关键字	$K1$	$K2$	$K3$	$K4$	$K5$	$K6$	$K7$	$K8$	$K9$	$K10$	$K11$	$K12$	$K13$

把当前查找区间的中间点位置上的元素作为根，左子表和右子表的元素分别作为根的左子树和右子树，由此得到的二叉树称为判断树。假定二叉树的深度为 d，那么满二叉树的节点数为 $2^0+2^1+2^2+\cdots+2^{d-1}=2^d-1$，树中第 i 层的元素个数为 2^{i-1}。通常，节点数为 n 的判定树不一定是满二叉树，但分叉数小于 2 的只限于最后一层，所以它的深度与满二叉树情况完全相同，即 $2^{d-1}-1<n\leqslant 2^d-1$，$d$ 为判定树深度。因此，n 个元素的判定树深度为

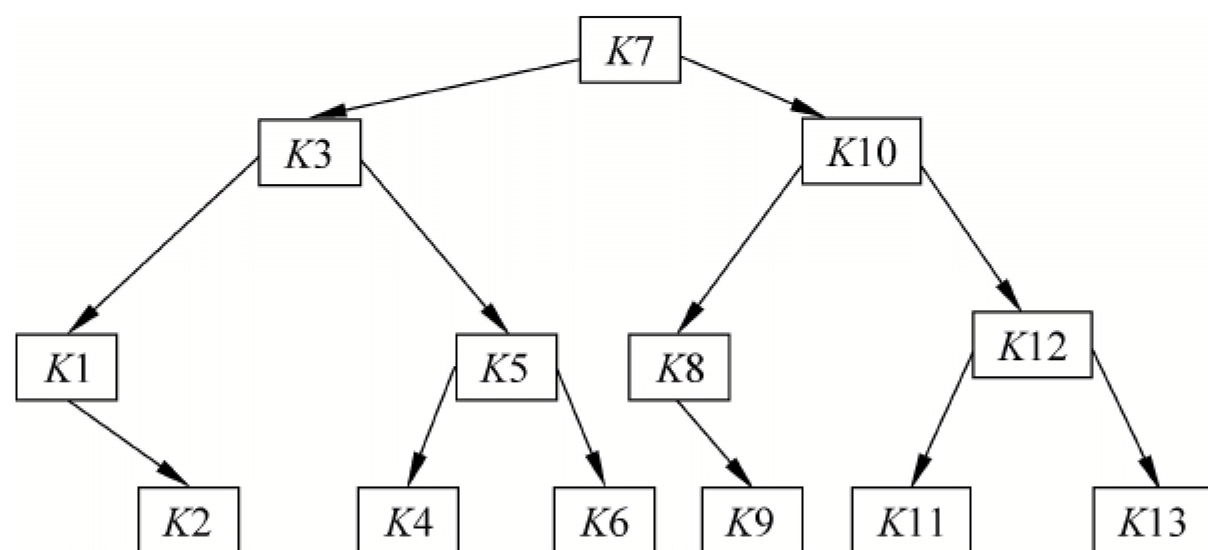


图 7-2-1　判定树

$\lceil \log_2(n+1) \rceil$，符号$\lceil\ \rceil$表示向上取整(类似于 Python 语言中 ceil 函数)。

注：等比数列 $a1 \cdots an$ 的求和公式为 $Sn = a1(1-q^n)/(1-q)$ 或 $Sn=(a1- an \cdot q)/(1-q)(q \neq 1)$，其中 q 为公比，n 为项数，$a1$ 为第一项，an 为最后一项。

查找过程即为从根节点开始比较，直到查找到该记录或者到叶子节点时才结束，所以查找过程所用的时间集中在经历每个节点时与该节点记录的比较。查找根节点比较次数为1；成功查找第 i 层每个记录比较的次数为 i，如上例中查找 $K5=32$ 的比较次数为 3 次；在最坏情况下，即被查找的值位于叶子节点或不存在，这时的比较次数为判定树深度，如上例中查找 $K4=25$ 的比较次数为 4 次，查找不存在的 28 的比较次数也为 4 次。当 n 足够大时，可记为 $O(\log_2 n)$。

由于每次循环需若干次比较和赋值操作，因此在最坏情况下，总的时间性能为 $O(\log_2 n)$。折半查找的效率是非常高的。

【例 7-2-2】 在一个包含两百万个人名的电话簿中找一个名字，折半查找可以不超过多少次就能找到指定的名字？

答案：21 次，因为 $\log_2 2\,000\,000 < 21$。

折半查找要求查找表按关键字有序，而排序是一种很费时的运算；另外，折半查找要求表是顺序存储的，为保持表的有序性，在进行插入和删除操作时，都必须移动大量记录。因此，折半查找的高查找效率是以牺牲排序为代价的，它特别适合于一经建立就很少移动、而又经常需要查找的线性表。

7.3　排序问题

在计算机科学中，排序是一个基本的操作，很多程序把它作为一个中间步骤，因而人们设计出了大量的排序算法。对于一个具体的应用，选择哪一种算法合适取决于待排序的元素个数，这些元素排序的程度，以及所使用的存储设备等。下面介绍几种常见的排序算法，每一种算法都是从解决思路、算法过程描述和算法性能分析等几个方面进行的。

7.3.1　冒泡排序

1. 问题的解决思路

冒泡排序方法是最简单的排序方法。这种方法的基本思想是：将待排序的元素看作竖着排列的“气泡”，较小的元素比较轻，从而要往上浮。冒泡排序算法要对这个“气泡”序列处

理若干遍。所谓一遍处理,就是自底向上检查一遍这个序列,并时刻注意两个相邻的元素的顺序是否正确。如果发现两个相邻元素的顺序不对,即"轻"的元素在下面,就交换它们的位置。显然,处理一遍之后,"最轻"的元素就浮到了最高位置;处理二遍之后,"次轻"的元素就浮到了次高位置。在做第二遍处理时,由于最高位置上的元素已是"最轻"元素,所以不必检查。一般地,第 i 遍处理时,不必检查第 i 高位置以上的元素,因为经过前面 $i-1$ 遍的处理,它们已正确地排好序。

【例 7-3-1】 序列[49 38 65 97 76 13 27 49]中的元素个数为 $n=8$,则最多做 $8-1=7$ 次循环,$i=1, 2, \cdots, n-1$。在第 i 遍中顺次两两比较 $A[j-1]$和 $A[j]$,$j=n-1, n-2, \cdots, i$。如果发生逆序,则交换 $A[j-1]$和 $A[j]$。

i = 01 02 03 04 05 06 07

49 13 13 13 13 13 13 13

38 49 27 27 27 27 27 27

65 38 49 38 38 38 38 38

97 65 38 49 49 49 49 49

76 97 65 49 49 49 49 49

13 76 97 65 65 65 65 65

27 27 76 97 76 76 76 76

49 49 49 76 97 97 97 97

2. 解决问题过程的描述

```
冒泡排序算法 BubbleSort。
//按照一定的计算步骤将一系列数排成升序
//输入: 一列数<a1, a2, …, an>,即实数数组 A[0:n−1]
//输出: 对输入数列的一个变换<a1′, a2′, …, an′>,其中 a1′≤a2′≤…≤an′,即升序排列的 A[0:n−1]
for i ← 1 to n−1 do
  for j ← n−1 to i do
    if A[j−1] > A[j]
        temp←A[j−1]  A[j−1]←A[j]  A[j]←temp
```

3. 算法性能的考虑

在最好的情形下,元素的初始排列已经按关键字从小到大排好序时,此算法只执行一遍起泡,做 $n-1$ 次关键字比较,不移动对象。

最坏的情形是算法执行了 $n-1$ 遍起泡,第 i 遍做了 $n-i$ 次关键字比较,执行了 $n-i$ 次对象交换。这样在最坏情形下总的关键字比较次数 KCN 和对象移动次数 RMN 为

$$\mathrm{KCN} = \sum_{i=1}^{n-1}(n-i) = \frac{1}{2}n(n-1)$$

$$\mathrm{RMN} = 3\sum_{i=1}^{n-1}(n-i) = \frac{3}{2}n(n-1)$$

因此,在最坏情形下冒泡算法的时间性能为 $O(n^2)$。

注:等差数列 $a1 \cdots an$ 的求和公式为 $Sn=na1+n(n-1)d/2(d\neq 0)$或 $Sn=n(a1+an)/$

2,其中 d 为公差,n 为项数,$a1$ 为第一项,an 为最后一项。

7.3.2 选择排序

1. 问题的解决思路

选择排序的基本思想是对待排序的记录序列进行 $n-1$ 遍的处理,第 i 遍处理是将 $A[i..n]$中最小者与 $A[i]$交换位置。这样,经过 i 遍处理之后,前 i 个记录的位置已经是正确的了。

【例 7-3-2】 选择排序示例。

初始关键字[49 38 65 97 76 13 27 49]

第一遍排序后 13 [38 65 97 76 49 27 49]

第二遍排序后 13 27 [65 97 76 49 38 49]

第三遍排序后 13 27 38 [97 76 49 65 49]

第四遍排序后 13 27 38 49 [49 97 65 76]

第五遍排序后 13 27 38 49 49 [97 65 76]

第六遍排序后 13 27 38 49 49 65 [76 97]

第七遍排序后 13 27 38 49 49 65 76 [97]

最后排序结果 13 27 38 49 49 65 76 97

2. 解决问题过程的描述

```
选择排序算法 SelectSort。
//按照一定的计算步骤将一系列数排成升序
//输入: 一列数<a1,a2,…,an>,即实数数组 A[0:n-1]
//输出: 对输入数列的一个变换<a1',a2',…,an'>,其中 a1'≤a2'≤…≤an',即升序排列的 A[0:n-1]
for i ← 1 to n-1 do
  min ← i
  for j ← i+1 to n do
    if A[j] < A[min]
        min ← j          //当前序列中的最小元素
    if min≠i             //对换到第 i 个位置
        temp ← A[i]  A[i]←A[min]  A[min]← temp
```

3. 算法性能的考虑

选择排序的关键字比较次数 KCN 与对象的初始排列无关,第 i 遍选择最小值对象所需的比较次数总是 $n-i-1$,其中,n 为整个待排序对象序列的元素个数。因此,总的关键字比较次数为

$$\mathrm{KCN}=\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}$$

对象的移动次数与对象序列的初始排列有关。当这组对象的初始状态是按其关键字从小到大有序排列的时候,对象的移动次数 RMN = 0,达到最少。

最坏情况是每一遍都要进行交换,总的对象移动次数为 RMN = $3(n-1)$。

因此,直接选择算法的时间性能为 $O(n^2)$。

7.3.3 插入排序

1. 问题的解决思路

直接插入排序的基本思想是当插入第 i 个对象时,前面的 $A[0]$,$A[1]$, …, $A[i-1]$已经排好序。这时,用 $A[i]$的关键字与 $A[i-1]$, $A[i-2]$, …的关键字顺序进行比较,找到插入位置即将 $A[i]$插入,原来位置上的对象向后顺移。

【例 7-3-3】 插入排序示例。

[初始关键字] [49] 38 65 97 76 13 27 49

j=2(38) [38 49] 65 97 76 13 27 49

j=3(65) [38 49 65] 97 76 13 27 49

j=4(97) [38 49 65 97] 76 13 27 49

j=5(76) [38 49 65 76 97] 13 27 49

j=6(13) [13 38 49 65 76 97] 27 49

j=7(27) [13 27 38 49 65 76 97] 49

j=8(49) [13 27 38 49 49 65 76 97]

2. 解决问题过程的描述

```
插入排序算法 InsertSort。
//按照一定的计算步骤将一系列数排成升序
//输入: 一列数<a1,a2,…,an>,即实数数组 A[0:n-1]
//输出: 对输入数列的一个变换<a1',a2',…,an'>,其中 a1'≤a2'≤…≤an',即升序排列的 A[0:n-1]
for i ← 1 to n-1 do
  temp ← A[i] j ← i                //从后向前顺序比较
  while temp < A[j-1] and j > 0 do
        A[j]←A[j-1] j←j-1          //将大于 A[i]的元素后移
  A[j] ← temp                      //插入 A[i]
```

3. 算法性能的考虑

关键字比较次数和对象移动次数与对象关键字的初始排列有关。最好情况下,排序前对象已经按关键字大小从小到大排列了,每遍只需与前面的有序对象序列的最后一个对象的关键字比较一次,移动两次对象,总的关键字比较次数为 $n-1$,对象移动次数为 $2(n-1)$。

最坏情况下,第 i 遍时第 i 个对象必须与前面 i 个对象都做关键字比较,并且每做一次比较就要做一次数据移动。则总的关键字比较次数 KCN 和对象移动次数 RMN 分别为

$$\mathrm{KCN} = \sum_{i=1}^{n-1} i = n(n-1)/2 \approx n^2/2,$$

$$\mathrm{RMN} = \sum_{i=1}^{n-1} (i+2) = (n+4)(n-1)/2 \approx n^2/2$$

因此,直接插入排序的时间性能为 $O(n^2)$。

7.3.4 快排-引入递归和分治概念

1. 问题的解决思路

快速排序(Quicksort)是由 C. A. R. Hoare(东尼·霍尔)在 1962 年提出的一种排序方法。快速排序算法的基本思想本身就是分治法。通过分割,将无序序列分成两部分,其中前一部分的元素值都小于后一部分的元素值。然后每一部分再各自递归进行上述过程,直到每一部分的长度为 1 为止。具体过程是在当前无序区 $A[1..n]$中任取一个数据元素 x 作为比较的"基准",用此基准将当前无序区划分为左右两个较小的无序区 $A[1..i-1]$和 $A[i+1..n]$,且左边的无序子区中数据元素均小于等于基准元素,右边的无序子区中数据元素均大于等于基准元素,而基准 x 则位于最终排序的位置上,即 $A[1..i-1]\leqslant x\leqslant A[i+1..n]$ $(1\leqslant i\leqslant n)$,当 $A[1..i-1]$和 $A[i+1..n]$均非空时,分别对它们进行上述的划分过程,直至所有无序子区中的数据元素均已排序为止。

【例 7-3-4】 快排示例。

初始关键字[49 38 65 97 76 13 27 49]

第一次划分过程:

x 的初始值为序列的第一个元素 49。

j 从右向左扫描,查找第一个小于 x 的元素为 27,第一次交换后为[27 38 65 97 76 13 49 49]。

i 从左向右扫描,查找第一个大于 x 的元素为 65,第二次交换后为[27 38 49 97 76 13 65 49]。

j 向左扫描,位置不变,第三次交换后为[27 38 13 97 76 49 65 49]。

i 向右扫描,位置不变,第四次交换后为[27 38 13 49 76 97 65 49]。

j 向左扫描,此时 $i=j$,第一遍排序结束时为[27 38 13 49 76 97 65 49]。

各遍排序之后的状态:

初始关键字为[49 38 65 97 76 13 27 49]。

一遍排序之后为[27 38 13] 49 [76 97 65 49]。

二遍排序之后为[13] 27 [38] 49 [49 65] 76 [97]。

三遍排序之后为 13 27 38 49 49 [65] 76 97。

最后的排序结果为 13 27 38 49 49 65 76 97。

2. 解决问题过程的描述

```
快速排序算法 QuickSort。
//按照一定的计算步骤将一系列数排成升序
//输入: 一列数<a1,a2,…,an>,即实数数组 A[0:n-1]
//输出: 对输入数列的一个变换<a1',a2',…,an'>,其中 a1'≤a2'≤…≤an',即升序排列的 A[0:n-1]
下面的过程(函数)实现了快速排序,为排序一个完整的数组 A,最初的调用是 QuickSort(A, 1, n)。
QuickSort(A, left, right)              //对 A[left..right]快速排序
if left < right                        //当 A[left..right]为空或只有一个元素时不需要排序
i ← Partion(A, left, right)            //对 A[left..right]做划分
QuickSort(A, left, i-1)                //递归处理左区间 A[left,i-1]
QuickSort(A, i+1, right)               //递归处理右区间 A[i+1.. right]

快速排序算法的关键是 Partion 函数,它对子数组 A[left..right]进行划分。
Partion(A, 1, h)
```

```
//对无序区 A[l,h]做划分,i给出本次划分后已被定位的基准元素的位置
i ← 1 j ← h x ← A[i]                  //初始化,x为基准
while i < j do
   while A[j] >= x and i < j do
      j ← j − 1                       //从右向左扫描,查找第一个小于 x 的元素
   if i < j                           //已找到 A[j] < x
      A[i] ← A[j] i ← i + 1          //相当于交换 A[i]和 A[j]
   while A[i] <= x and i < j do
      i ← i + 1                       //从左向右扫描,查找第一个大于 x 的元素
   if i < j                           //已找到 A[i] > x
      A[j] ← A[i] j ← j − 1          //相当于交换 A[i]和 A[j]
A[i] ← x                              //基准 x 已被最终定位
return i                              //最后基准对象安放到位,函数返回其位置
```

3. 算法性能的考虑

基准元素选取的不确定性以及待排序记录初始顺序的随机性,均会对排序效率产生影响。以下分别对最坏情况、最好情况、平均情况下快速排序的时间性能来加以分析,其中$n>1$,c为常数。

1) 最坏情况分析

此时基准元素始终是最小元素,那么对n个元素的序列进行排序所需的时间$T(n)$的递推关系为

$$
\begin{aligned}
T(n) &= T(n-1)+cn \\
T(n-1) &= T(n-2)+c(n-1) \\
T(n-2) &= T(n-3)+c(n-2) \\
&\vdots \\
T(2) &= T(1)+c(2)
\end{aligned}
$$

将上述等式左右两边分别相加,可得

$$
\begin{aligned}
T(n) &= T(1)+c(2+3+4+5+6+\cdots+n) \\
&= T(1)+c(n-1)(n+2)/2
\end{aligned}
$$

因此,时间性能为

$$T(n) = O(n^2)$$

2) 最好情况分析

此时基准元素正好位于中间,即每次划分对一个对象定位后,该对象的左侧子序列与右侧子序列的长度相同,下一步将是对两个长度减半的子序列进行排序,则

$$T(n) = 2T(n/2)+cn$$

上式两边同除以n,并递推可得

$$
\begin{aligned}
T(n)/n &= T(n/2)/(n/2)+c \\
T(n/2)/(n/2) &= T(n/4)/(n/4)+c \\
T(n/4)/(n/4) &= T(n/8)/(n/8)+c \\
&\vdots \\
T(2)/2 &= T(1)+c\log_2 n
\end{aligned}
$$

将上述等式左右两边分别相加得

$$T(n) = cn\log_2 n + n = O(n\log_2 n)$$

3) 平均情况分析

假设对于 A,每一个区间的大小都是等可能的,即概率为 $1/n$,

$T(i)$($T(n-i-1)$)的平均值为$(1/n)\sum_{i=0}^{n-1}T(i)$。

又快速排序的时间等于两个递归调用的时间加上花费在分割上的线型时间,可得

$$\begin{aligned} T(n) &= T(i) + T(n-i+1) + cn \\ &= 2/n[0+1+2+3+\cdots+(n-1)] + cn \end{aligned}$$

上式两边同乘以 n,并递推可得

$$nT(n) = 2[0+1+2+3+\cdots+(n-1)] + cn^2$$
$$(n-1)T(n-1) = 2[0+1+2+3+4+\cdots+(n-2)] + c(n-1)^2$$

两式相减,可得

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c$$
$$nT(n) = (n+1)T(n-1) + 2cn$$

递推可得

$$T(n)/(n+1) = T(n-1)/(n) + 2c/(n+1)$$
$$T(n-1)/(n) = T(n-2)/(n-1) + 2c/(n)$$
$$T(n-2)/(n-1) = T(n-3)/(n-2) + 2c/(n-1)$$
$$T(2)/3 = T(1)/2 + 2c/3$$

将上述等式两边分别相加,可得

$$\begin{aligned} T(n)/(n+1) &= T(1)/2 + 2c[1/3 + 1/4 + 1/5 + \cdots + 1/(n+1)] \\ &= O(\log_2 n) \end{aligned}$$

因此,平均时间性能为

$$T(n) = O(n\log_2 n)$$

就 n 较大的平均计算时间而言,快速排序是我们所讨论的所有排序方法中最好的一个。

7.4 递归和分治的思想

当求解的问题规模较大时,往往会采用分解的方法,将大问题转换为小问题,分而治之;对于同类型的问题,还可采用递归法由小问题的解构造出大问题的解。

7.4.1 递归概念

"你站在桥上看风景,看风景人在楼上看你,明月装饰了你的窗子,你装饰了别人的梦。"——卞之琳

在日常生活中递归一词常用于描述以自相似方法重复事物的过程。例如,这首诗就包含了一个递归的概念。在计算机科学中,递归(recursion)是一个十分重要的概念,是求解问题的常用方法。它指一种通过重复将问题分解为同类子问题而解决问题的方法。重复是机器最擅长的事了。

在介绍递归调用之前,我们先来看一道数学题。进而分析并引出斐波那契数列——它

是递归调用中非常典型的一个案例。

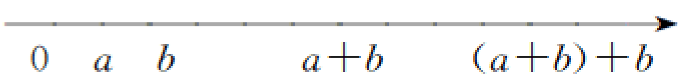
【例 7-4-1】 从 1,2,3,…,2014 中最少应选出多少个不同的数,才能保证选出的数中必存在三个不同的数构成一个三角形的三边长?

(1) 分析。

由于形成三角形的充要条件是任何两边之和大于第三边,因此不构成三角形的条件就是任意两边之和不超过最大边。设最少应选出 n 个数,才能保证选出的数中必存在三个不同的数构成一个三角形的三边长,则需要证明选出最多的 $n-1$ 个数不能构成三角形。

(2) 证明。

取出 $n-1$ 个数无法构成三角形,即 $c \geqslant a+b$,其中 a、b、c 为三角形的三边。那么如何选数可使 $n-1$ 最大?分析可知当满足边界条件 $c=a+b$ 时,即下图中点排列最紧密时,取得数最多。所以取出 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597 共 16 个数,此时再取出任意一个其他数必落在两数之间,与其左边两个数必能构成三角形,所以 n 为 17。



(3) 讨论。

因此,这道数学题就转化为了求斐波那契数列中小于等于 2014 的各项。当然像前面那样死算是可以的,留到后面通过编程来实现(实验范例 4)。

递归是设计和描述算法的一种有力的工具,简单地说,递归就是用自己来定义自己。能采用递归描述的算法通常有这样的特征:为求解规模为 N 的问题,设法将它分解成规模较小的问题,然后从这些小问题的解构造出大问题的解,并且这些规模较小的问题也能采用同样的分解和综合方法,分解成规模更小的问题,并从这些更小问题的解构造出规模较大问题的解。特别地,当规模 $N=1$ 时,能直接得解。

【例 7-4-2】 编写计算斐波那契(Fibonacci)数列第 n 项的函数 fib(n)。

(1) 问题描述。

西方最先研究这个数列的是意大利数学家 Fibonacci,他描述理想状态下兔子生长数目时用了该数列。

兔子在出生两个月后,就有繁殖能力,一对兔子每个月能生出一对小兔子来。如果所有兔都不死,那么一年以后可以繁殖多少对兔子?

第一个月初有一对刚诞生的雌雄兔子(初始原点)。

第三个月初它们可以生育。

每月每对可生育的兔子会诞生下一对新兔子。

假设在 n 月有可生育的兔子总共 $f(n)$ 对,那么 $f(n)=f(n-1)+f(n-2)$;因为在 n 月的时候,前一月($n-1$ 月)的 $f(n-1)$ 对兔子可以存留至第 n 月(在当月属于新诞生的兔子尚不能生育)。而新生育出的兔子对数等于所有在 $n-2$ 月就已存在的 $f(n-2)$ 对。

这就是斐波那契数列的递归定义:fib(0)=0;fib(1)=1;fib(n)=fib($n-1$)+fib($n-2$)(当 $n>1$ 时)。斐波那契数列为{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …}。

(2) 解决问题思路。

递归算法的执行过程分递推和回归两个阶段。在递推阶段,把较复杂的问题(规模

为 n)的求解推到比原问题简单一些的问题(规模小于 n)的求解。例如求解 fib(n),把它推到求解 fib($n-1$)和 fib($n-2$)。也就是说,为计算 fib(n),必须先计算 fib($n-1$)和 fib($n-2$),而计算 fib($n-1$)和 fib($n-2$),又必须先计算 fib($n-3$)和 fib($n-4$)。以此类推,直至计算 fib(1)和 fib(0),分别能立即得到结果 1 和 0。在递推阶段,必须要有终止递归的情况。例如在函数 fib 中,当 n 为 1 和 0 的情况。在回归阶段,当获得最简单情况的解后,逐级返回,依次得到稍复杂问题的解,得到 fib(1)和 fib(0)后,返回得到 fib(2)的结果,……,在得到了 fib($n-1$)和 fib($n-2$)的结果后,返回得到 fib(n)的结果。

由于递归引起一系列的函数调用,并且可能会有一系列的重复计算,递归算法的执行效率相对较低。当某个递归算法能较方便地转换成递推算法时,通常按递推算法编写程序。例如计算斐波那契数列第 n 项的函数 fib(n)应采用递推算法,即从斐波那契数列的前两项出发,逐次由前两项计算出下一项,直至计算出要求的第 n 项。详见 7.7.3 节实验内容(5):非递归的循环方式计算斐波那契数列的第 n 项。

(3) 过程描述。

```
fibonacci(n)        // n 为斐波那契数列的第 n 个元素
if n = 0 or n = 1:
return n
else
return fibonacci(n-1) + fibonacci(n-2)
```

(4) 算法性能的考虑。

当 n=0 时,T(n)=1

当 n=1 时,T(n)=1

当 n>1 时

$T(n)= T(n-1)+ T(n-2)$

$\leqslant 2T(n-1) \leqslant 2^2T(n-2) \leqslant \cdots \leqslant 2^{n-1}T(1)$

$=O(2^n)$

7.4.2 递归调用方法与实现

递归算法既是一种有效的算法设计方法,也是一种有效的分析问题的方法。递归算法求解问题的基本思想是对于一个较为复杂的问题,把原问题分解成若干个相对简单且类同的子问题,而简单到一定程度的子问题可以直接求解。这样,原问题就可递推得到解。

递归算法是通过子程序或函数来实现的。

对于非递归函数,调用函数在调用被调用函数前,系统要保存以下两类信息:

(1) 调用函数的返回地址(从而能执行下一语句);

(2) 调用函数的局部变量值。

当执行完被调用函数,返回调用函数前,系统首先要恢复调用函数的局部变量值,然后返回调用函数的返回地址。

递归函数被调用时,系统要做的工作和非递归函数被调用时系统要做的工作在形式上雷同,但保存信息的内容和方法不同。

1. 保存内容

每一层递归调用所需要保存的信息构成一个工作记录,通常包括以下内容:

(1) 本次递归调用中的局部变量值;

(2) 返回地址,即本次递归过程调用语句的后继语句的地址;

(3) 本次调用中与形参结合的实参值,包括函数名、引用参数与数值参数等。

2. 保存方法

递归函数被调用时,系统在运行递归函数前也要保存调用函数的返回地址和局部变量。但因为递归函数的运行特点,是最后被调用的函数要最先被返回,若按非递归函数那样保存信息,显然要出错。

由于堆栈(一种特定的数据结构)的后进先出特性正好与递归函数调用和返回的过程吻合,因此,高级程序设计语言利用堆栈保存递归函数调用的信息,系统用于保存递归函数调用信息的堆栈称为运行时栈(表 7-4-1)。

表 7-4-1　运行时栈示意

栈顶	局部变量 n	返回地址 n	参数 n
	⋮	⋮	⋮
	局部变量 2	返回地址 2	参数 2
栈底	局部变量 1	返回地址 1	参数 1

递归函数被调用时,在每进入下一层递归调用时,系统就建立一个新的工作记录,并把这个工作记录放进栈内成为运行时栈新的栈顶;每返回一层递归调用,就退栈一个工作记录。

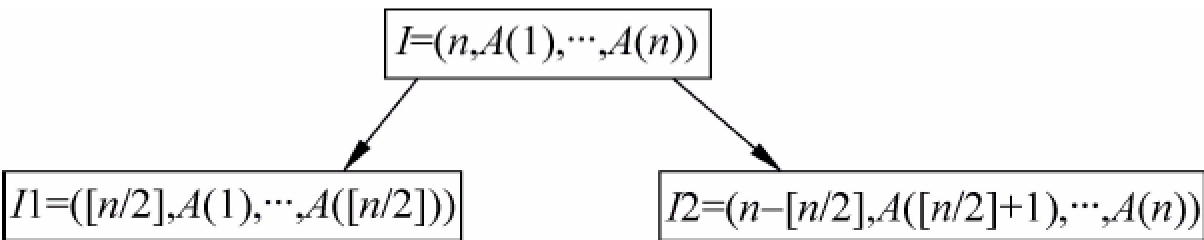
7.4.3 分治概念

上面的快速排序以及前面介绍的折半查找算法贯彻一个思想,即分治法。当人们要解决一个输入规模 n 很大的问题时,往往会想到将该问题分解。例如将这 n 个输入分成 k 个不同的子集。如果能得到 k 个不同的可独立求解的子问题,而且在求出这些子问题的解之后,还可以找到适当的方法把它们的解合并成整个问题的解,那么复杂的难以解决的问题就可以得到解决。这种将整个问题分解成若干个小问题来处理的方法称为分治法。一般来说,被分解出来的子问题应与原问题具有相同的类型,这样便于算法实现(多数情况下采用递归算法)。如果得到的子问题相对来说还较大,则再用分治法,直到产生出不用再分解就可求解的子问题为止。人们考虑和使用较多的是 $k=2$ 的情形,即将整个问题二分。

【例 7-4-3】 采用分治法求 n 元数组中的最大和最小元素。

(1) 解决问题思路。

将任一实例 $I=(n,A(1),\cdots,A(n))$ 分成一些较小的实例来处理。



(2) 过程描述。

```
MaxMin(i,j,fmax,fmin)            //A[1:n]是 n 个元素的数组,参数 i,j
//是整数,1≤i≤j≤n,使用该过程将数组 A[i..j]中的最大最小元素
//分别赋给 fmax 和 fmin。
if i=j
fmax← A[i] fmin← A[i];           //子数组 A[i..j]中只有一个元素
else if i=j-1                    //子数组 A[i..j]中只有两个元素
  if A[i]< A[j]
     fmin← A[i] fmax← A[j]
  else fmin← A[j] fmax← A[i]
else
mid←(i+j)/2                      //子数组 A[i..j]中的元素多于两个
MaxMin(i, mid, lmax, lmin)       //递归调用部分
MaxMin(mid+1, j, rmax, rmin)
fmax← max(lmax, rmax)
fmin← min(lmin, rmin)
```

(3) 算法性能的考虑。

MaxMin 的元素比较次数。

当 $n=1$ 时,$T(n)=0$;

当 $n=2$ 时,$T(n)=1$。

当 $n=2^k$(k 是某个正整数)> 2 时,

$$
\begin{aligned}
T(n) &= 2T(n/2)+2 \\
&= 4T(n/4)+4+2 \\
&\cdots\cdots \\
&= 2^{k-1}T(2)+(2+2^2+\cdots+2^i+\cdots+2^{k-1}) \\
&= 2^{k-1}+2^k-2 \\
&= 3n/2-2
\end{aligned}
$$

7.5 本章小结

本章重点介绍了算法性能分析方法、经典的查找排序算法、计算机求解中常用的递归分治方法以及性能分析在这些算法中的应用。

本章要点如下:

(1) 算法性能分析又称为算法复杂性分析,是研究算法效率的一种理论分析方法。它广泛应用于计算机科学领域,用于评估算法的可行性、不同算法的优劣以及程序运行的效率。

(2) 查找又称为检索,是一种十分有用的操作。查找与数据的存储结构和组织方式有关,对于顺序存放的数据介绍了顺序查找和折半查找两种方式,而后者要求数据是有序排列的。查找算法的性能分析通常用查找过程中对关键字的比较次数来衡量。

(3) 排序就是整理数据使之按关键字递增或递减的次序排列。冒泡排序的基本思想是：通过比较和交换相邻元素，使关键字最小的元素总是“漂浮”在最上面。选择排序的基本思想是：每一次从待排序的元素中选出关键字最小的元素，顺序放在排好序的子列表最后。插入排序的基本思想是：每次将一个待排序的元素，按其关键字的大小插入已排序的子列表中的适当位置。所有排序算法的最坏时间复杂度都为 $O(n^2)$，但快排的平均时间复杂度为 $O(n\log_2 n)$。

(4) 在算法设计中，递归是常用的求解方法。在定义一个过程或函数时，若出现了对本过程或函数的调用，则称为递归。使用递归方法常常因为问题的定义是递归的，例如斐波那契数列；或者问题的求解方法是递归的，例如汉诺塔问题。

(5) 分治法又称为分而治之法，它是一种将整个大问题分解成若干个小问题来处理的方法。一般被分解出来的子问题应与原问题具有相同的类型，这样便于算法实现。

7.6 习题与思考

7-1 按照渐近阶，从低到高的顺序排列下列表达式：

$n^{2/3}$，$n!$，$6n^2$，$\log_2 n$，3^n，10，$18n$，2^n

7-2 请思考：以下的查找最大最小值算法，如果将语句 if $A[i] < \text{minval}$ 改为 else if $A[i] < \text{minval}$，算法的基本操作次数会有变化吗？

```
算法 MaxMinElement
//求给定数组中的最大最小元素
//输入：实数数组 A[0:n-1]
//输出：A 中的最大元素 maxval 和最小元素 minval
maxval ← A[0]
minval ← A[0]
for i ← 1 to n-1 do
    if A[i] > maxval
        maxval ← A[i]
    if A[i] < minval
        minval ← A[i]
```

7-3 折半查找平均情况下不成功检索的时间性能为________。

7-4 冒泡排序与选择排序在一般情况下哪一个效率更高些？

7-5 请分别用冒泡、选择、插入和快速排序法升序排列下面的实例，给出每一趟排序的结果。

(6，22，9，10，4，34，27，19，15，6)

7-6 请推导快速排序算法的时间性能。

7-7 分治法的基本思想是将一个规模为 n 的问题分解为与原问题________(相同/不相同)的 k 个规模较小且________(互相独立/相关)的子问题。

7-8 一个直接或间接地调用自身的算法称为________，它有两个条件，一个是要直接或间接地调用自身，另一个是必须有________。

7.7 实验 算法实现与性能分析

7.7.1 实验目标

(1) 综合应用已学的程序设计方法及各种控制语句。
(2) 进一步熟练掌握程序的编写与调试。
(3) 掌握几种特定数的查找算法。
(4) 掌握冒泡排序、选择排序、插入排序的算法与编程实现。
(5) 掌握递归的概念和编程方法。
(6) 围绕几种典型的算法,学会时间性能分析的方法。

7.7.2 实验范例

1. 折半查找法

1) 问题描述

对于有序的关键字序列[5,7,13,25,32,46,54,62,78,83,88,91,99],使用折半查找法编程搜索值为32的关键字所在的位置。

2) 算法设计

如果不是从一组随机的序列里查找,而是从一组排好序的序列里找出某个元素的位置,则可以有更快的算法。由于这个序列已经从小到大排好序了,每次取中间的元素和待查找的元素比较,如果中间的元素比待查找的元素大,就说明"如果待查找的元素存在,一定位于序列的前半部分",这样可以把搜索范围缩小到前半部分,然后再次使用这种算法迭代。这种"每次将搜索范围缩小一半"的思想称为Binary Search。

3) 程序实现

```
def BinarySearch(a, target):
    left = 0
    right = len(a) - 1
    while left <= right:
        mid = (left + right)// 2
        midVal = a[mid]
        if midVal < target:
            left = mid + 1
        elif midVal > target:
            right = mid - 1
        else:
            return mid
    return -1
L = [5,7,13,25,32, 46, 54, 62, 78, 83, 88, 91, 99]
print(BinarySearch(L,32))
```

4) 执行结果

```
>>>
4
>>>
```

5）分析与思考

这个算法容易出错的地方很多，例如 mid ＝（left ＋ right）// 2 这一句，在数学概念上其实是 mid $=\lfloor (left + right)/2 \rfloor$，还有 left ＝ mid ＋ 1 和 right ＝ mid－1，如果前者写成了 left ＝ mid；或后者写成了 right ＝ mid；那么很可能会导致死循环（想一想什么情况下会死循环）。

请思考：这个算法的时间性能。

2. 冒泡排序法

1）问题描述

对于序列［49，38，65，97，76，13，27，49］，使用冒泡排序法编程实现从小到大的排列。

2）算法设计

冒泡排序算法的执行过程是先比较相邻的元素，如果前一个比后一个大，就交换这两个数；对每一对相邻元素做同样的操作，从开始到结尾，因此最后的元素会是最大的数；除去最后一个，针对所有的元素再重复以上步骤；持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

3）程序实现

```
#冒泡排序算法
def bubble(List):
    for j in range(len(List) - 1,0, - 1):
        print(List)
        for i in range(0,j):
            if List[i]> List[i + 1]:List[i],List[i + 1] = List[i + 1],List[i]
    return List
#初始化输入数据
testlist = [49, 38, 65, 97, 76, 13, 27, 49]
print('结果:', bubble(testlist))
```

4）运行结果

```
>>>
[49, 38, 65, 97, 76, 13, 27, 49]
[38, 49, 65, 76, 13, 27, 49, 97]
[38, 49, 65, 13, 27, 49, 76, 97]
[38, 49, 13, 27, 49, 65, 76, 97]
[38, 13, 27, 49, 49, 65, 76, 97]
[13, 27, 38, 49, 49, 65, 76, 97]
[13, 27, 38, 49, 49, 65, 76, 97]
结果: [13, 27, 38, 49, 49, 65, 76, 97]
>>>
```

5）分析与思考

冒泡排序的实现不考虑序列是否已经排序好，所以它的执行效率为 $O(n^2)$，如果能在内部循环第一次执行时，使用标志来表示有无交换的必要，有可能把最好的时间性能降低到 $O(n)$。在这种情况下，已经排序好的序列就可以不再做交换了。

3. 插入排序法

1）问题描述

对于序列[31,45,35,56,37,69,310,21,12]，使用插入排序法编程实现从小到大的排列。

2）算法设计

插入排序算法类似于玩扑克时抓牌的过程，玩家只要保持手上的牌的顺序是正确的，每次抓到新的牌均按照顺序插入手上牌的中间。那么无论抓了几张牌，最后手上的牌都是依照顺序排列的。编程对一个数组进行插入排序也是同样的道理，但和插入扑克牌有一点不同，不可能在两个相邻的存储单元之间再插入一个单元，因此要将插入点之后的数据依次往后移动一个单元。这种算法在排完前 j 个数之后，可以保证 $A[0..j-1]$是局部有序的，保证了插入过程的正确性。为了更清楚地观察排序过程，在每次循环开头插了打印语句，在排序结束后也插了打印语句。

3）程序实现

```
#插入排序算法
def InsertionSort(A):
    for j in range(1,len(A)):
        print(A)
        key = A[j]
        i = j-1
        #向前查找插入位置
        while i>=0 and A[i]>key:
            A[i+1] = A[i]
            i = i-1
        A[i+1] = key
    return A
#初始化输入数据
A = [31,45,35,56,37,69,310,21,12]
print(InsertionSort(A))
```

4）执行结果

```
>>>
[31, 45, 35, 56, 37, 69, 310, 21, 12]
[31, 45, 35, 56, 37, 69, 310, 21, 12]
[31, 35, 45, 56, 37, 69, 310, 21, 12]
[31, 35, 45, 56, 37, 69, 310, 21, 12]
[31, 35, 37, 45, 56, 69, 310, 21, 12]
[31, 35, 37, 45, 56, 69, 310, 21, 12]
[31, 35, 37, 45, 56, 69, 310, 21, 12]
[21, 31, 35, 37, 45, 56, 69, 310, 12]
[12, 21, 31, 35, 37, 45, 56, 69, 310]
>>>
```

5）分析与思考

第一次执行循环之前，$j=1$，子序列 $A[0..j-1]$只有一个元素 $A[0]$，只有一个元素的

序列显然是排好序的；第 j 次循环之前，如果“子序列 $A[0..j-1]$是排好序的”这个前提成立，现在要把 key=$A[j]$插进去，按照该算法的步骤，把 $A[j-1]$、$A[j-2]$、$A[j-3]$等比 key 大的元素都依次往后移一个，直到找到合适的位置给 key 插入，就能证明循环结束时子序列 $A[0..j]$是排好序的。就像插扑克牌一样，“手中已有的牌是排好序的”这个前提很重要，如果没有这个前提，就不能证明再插一张牌之后也是排好序的；当循环结束时，$j=$ len(A)-1，如果“子序列 $A[0..j-1]$是排好序的”这个前提成立，那就是说 $A[0..\text{len}(A)-1]$是排好序的，也就是说整个数组 A 的所有元素都排好序了。

请思考：*这个算法的时间性能，比冒泡排序法好在哪里？*

4. 斐波那契序列应用

1）问题描述

从 1,2,3,…,2014 中最少应选出多少个不同的数，才能保证选出的数中必存在三个不同的数构成一个三角形的三边长？

2）算法设计

根据前面的分析可知，题目的要求是编程生成所需的斐波那契序列：该序列的最后一项是大于等于 2014 的最小整数。

3）程序实现

```
#count 用来记录最少应选出多少个不同的数
def fib(n):
  a,b = 1,1
  count = 0
  while b <= n:
    print(b, end = ',')
    a, b = b, a + b
    count += 1
  print(count + 1)

fib(2014)
```

4）执行结果

```
>>>
1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,17
>>>
```

5）分析与思考

这里采用了非递归的循环方式计算斐波那契数列的各项并打印输出。

请思考：*为什么不用递归算法？在什么情况下可以不用递归算法实现？*

5. 递归算法：Hanoi 塔问题

1）问题描述

这是印度的一个古老传说。一块黄铜板上插着三根宝石针，其中一根针上从下到上地穿好了由大到小的 64 片金片，即所谓的汉诺塔。游戏的目标是把所有金片从第一根针移到

第三根针上,第二根针作为中间过渡。每次只能移动一个金片,并且大的金片不能压在小的金片上面。

2) 算法设计

游戏中金片移动是一个很烦琐的过程。经计算,对于 64 个金片至少需要移动 $2^{64}-1\approx 1.8\times10^{19}$ 次(每秒移 1 次,4000 多亿年)。

不妨用 A 表示被移动金片所在的针(源),C 表示目的针,B 表示过渡针。对于把 $n(n>1)$ 个金片从第一根针 A 上移到第三根针 C 的问题可以分解成以下步骤:

① 将 $n-1$ 个金片从 A 经过 C 移动到 B。

② 将第 n 个金片移动到 C。

③ 再将 $n-1$ 个金片从 B 经过 A 移动到 C。

这样就把移动 n 个金片的问题转化为移动 $n-1$ 个金片的问题,即移动 n 个金片的问题可用移动 $n-1$ 个金片的问题递归描述,以此类推,可转化为移动一个金片的问题。显然,一个金片就可以直接移动。

3) 程序实现

```
# Hanoi 塔问题的递归实现
#count 用来记录移动次数
count = 1
def test(num, src, dst, rest):
        global count
        if num < 1:
                print(False)
        elif num == 1:
                print("%d:\t%s -> %s" % (count, src, dst))
                count += 1
        elif num > 1:
                test(num-1, src, rest, dst)
                test(1, src, dst, rest)
                test(num-1, rest, dst, src)
test(3, 'A', 'C', 'B')
```

4) 执行结果

```
>>>
1:A ->C
2:A ->B
3:C ->B
4:A ->C
5:B ->A
6:B ->C
7:A ->C
>>>
```

5) 分析与思考

假设 move 所需的时间为 1,则这个算法的时间性能。

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 = 2[2T(n-2)+1]+1 \\
&= 2^2T(n-2)+2+1 = 2^3T(n-3)+2^2+2+1 \\
&= \cdots \\
&= 2^{n-1}T(1)+2^{n-2}+\cdots+2^3+2^2+2+1 \\
&= 2^{n-1}+2^{n-2}+\cdots+2^3+2^2+2+1 \\
&= 2^n-1
\end{aligned}
$$

请计算：若有 64 个圆盘，则需要移动多少次？

7.7.3 实验内容

(1) 从一组数据中找出最大最小值。

提示：

① 示例的一组数据为 lst=[31,45,35,56,37,69,310,21,12]。

② range 类型常在 for 循环结构中使用，基本形式为 range([start]，stop[，step])，其中整数 start 可省略，缺省值为 0；整数 step 也可省略，缺省值为 1。常用的 range(start，stop)的取值范围是[start，stop)，即从 start 开始，每次加 1，最后的数等于 stop−1。

(2) 如果待查找的元素在序列中有多个，范例 1 的折半查找算法返回的是其中的任意一个，以[32，32，32，32，32，46，54，62，78，83，88，91，99]为例，如果调用 BinarySearch(32)则返回 2，即 $a[2]$，而有些场合下要求返回 $a[0]$，也就是说，如果待查找的元素在数组中有多个则返回第一个。请修改 7.7.2 实验范例 1 的折半查找算法实现这一特性。

(3) 补充程序空白处的语句，采用选择排序法将一组数据从小到大排列。

提示：

① 示例的一组数据为 testlist=[49，38，49，97，76，13，27，65]。

② 以下程序首先在未排序序列中找到最小元素，存放到排序序列 L 的起始位置 min_index=0，然后，再从剩余未排序元素中继续寻找最小元素，然后放到已排序序列的末尾 min_index 位置。以此类推，直到所有元素均排序完毕。对 n 个元素的列表进行排序至多进行 $n-1$ 次交换。

程序如下：

```
#选择排序算法
def selection_sort(L):
    N = len(L)
    exchanges_count = 0
    for i in range(________,N - 1):
        min_index = i
        for j in range(i + 1, N):
            if L[min_index] > L[j]:
                min_index = ________
        #若某个元素位于最终位置上，则它不会被移动，此时 min_index == i
        if min_index!= i:
            L[min_index], L[i] = L[i], L[min_index]
            exchanges_count += 1
        print('iteration #{}: {}'.format(i, L))
    print('Total {} swappings'.format(exchanges_count))
    return L
```

```
testlist = [49, 38, 49, 97, 76, 13, 27, 65]
print('Before selection sort: {}'.format(testlist))
print('After selection sort: {}'.format(selection_sort(testlist)))
```

执行结果：

```
>>>
Before selection sort: [49, 38, 49, 97, 76, 13, 27, 65]
iteration #0: [13, 38, 49, 97, 76, 49, 27, 65]
iteration #1: [13, 27, 49, 97, 76, 49, 38, 65]
iteration #2: [13, 27, 38, 97, 76, 49, 49, 65]
iteration #3: [13, 27, 38, 49, 76, 97, 49, 65]
iteration #4: [13, 27, 38, 49, 49, 97, 76, 65]
iteration #5: [13, 27, 38, 49, 49, 65, 76, 97]
iteration #6: [13, 27, 38, 49, 49, 65, 76, 97]
Total 6 swappings
After selection sort: [13, 27, 38, 49, 49, 65, 76, 97]
>>>
```

(4) 已知一个升序排列的整数序列[5,7,13,25,32, 46, 54, 62, 78, 83, 88, 91, 99]，现输入一个整数 x，要求按原来的规律将它插入这个整数序列中。

提示：

① 首先将整数序列保存到一个 list 中，用二分查找算法判断整数 x 在有序的整数序列中插入的位置。若 x 大于序列中最后一个数，则直接将 x 添加到序列中即可；若插入位置在中间，则先将插入位置所在及其之后的元素依次后移一个位置，然后将 x 插入相应位置处。

② 将插入位置所在及其之后的元素依次后移一个位置，可以利用一个循环语句，从最后一个元素开始，到插入位置所在的元素，把每个元素值依次赋值给其后面的元素即可。

(5) 补充以下程序空白处的语句，分别采用递归算法、改进的递归调用方法和非递归的循环方式计算斐波那契数列的第 n 项，并思考三种方法的执行效率。

提示：

① 首先分析一般递归调用方法为何效率低，看能否去除函数的重复计算，一种想法是利用 Python 中的数据结构，如列表和字典等，将计算过的函数保存起来；另一种想法是如有可能将递归转换成递推算法，编写程序。所谓递推，是指从已知的初始条件出发，逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定，或是通过对问题的分析与化简而确定。

② Python 的字典是由 key:value 对组成的数据结构，可以根据 key(索引)存储和得到相应的 value。

③ 计算斐波那契数列第 n 项的递推算法：从斐波那契数列的前两项出发，逐次由前两项计算出下一项，直至计算出要求的第 n 项。

程序如下：

```
#一般的递归算法
count = 0
```

```
def fibonacci(n):
    global count
    count += 1
    if n == 0 or n == 1:
        return ________
    else:
        return ________
print(fibonacci(5))
print(count)

#改进的递归调用算法
#第一句是定义字典并完成初始化
previous = {0:0, 1:1}
count = 0
def fibonacci_s(n):
    global count
    #可以用关键字 in 检查字典中是否有某个 key
    if n in previous:
    #previous[n]中保存的是 key 为 n 时的 value
        return previous[n]
    else:
        count += 1
        newValue = ________
        previous[n] = ________
        return newValue
print(fibonacci_s(5))
print(count)

#非递归的循环方式
def fib(n):
    #使用 list 存放斐波那契数列
    ret = []
    a,b = 0,1
    ret.append(a)
    for i in range(________,n + 1):
            ret.append(b)
            ________
    return ret
print(fib(5))
```

第8章　面向对象思想简介

计算机问题求解的过程是借助于计算机将所关心的现实世界映射到计算机世界的过程，此过程通常为：在正确认识现实世界（问题域）的基础上，借助某种建模思想建立相关模型，此后根据所建模型使用某种程序语言编程，再交由计算机执行求解。

目前计算机程序设计方法主要有两类：一类是面向过程；另一类是面向对象。在面向过程方法中，在正确分析问题域的基础上，借助流程图等手段建模并使用面向过程语言编程进行问题求解；在面向对象方法中，在正确认识现实世界的基础上，借助面向对象思想建模，并采用面向对象语言编程进行问题求解。

面向对象是一种思想、一种方法，它力求更客观更自然地描述现实世界。面向对象更符合人们的认知习惯，可使计算机世界更接近现实世界。

8.1　面向对象思想

8.1.1　面向对象思想概述

自20世纪80年代以来，面向对象思想已深入计算机问题求解领域的各个方面。它不只是解决具体问题的某种软件开发技术，而是关于如何看待计算机世界与现实世界之间的关系、用什么观点来研究问题并进行问题求解的思想方法。它力求更客观自然地描述现实世界，使分析、设计和系统实现的方法同人们认识客观世界的自然思维方式尽可能一致。

1. 面向过程和面向对象的区别

在传统的面向过程方法中，系统设计是围绕结构化体系模型进行的：首先将需要求解的问题域视为等待处理的一个大过程，然后进行功能分析，将系统（大过程）分解为若干个子功能模块（子处理过程）。期间，根据问题的复杂程度，子处理过程又可细化成更小的小过程，直至整个系统被分解为一个个易于处理的细分过程为止，之后再解决系统的总体控制问题。在传统开发方法中，数据与过程常常分离，缺乏对问题基本组成元素的完整分析，也欠缺灵活性，尤其是当需求功能变化时，将导致大量修改，不易维护。

为了克服传统开发方法的不足，面向对象方法解决问题的思路是从现实世界中客观存在的事物入手，强调直接以问题域（现实世界）中的客观事物为中心来认识问题、分析问题，并根据这些事物的本质特征，把它们抽象地表示为计算机系统中的对象，把对象作为系统的基本构成单位，又通过将对象之间的相互作用、相互联系映射到计算机系统来模拟现实客观世界。

2. 面向对象的主要优点

较之传统的面向过程的方法，面向对象具有以下主要优点：

1）自然高效

面向对象方法运用人们认识客观世界的自然思维方式来处理问题，使得软件开发者对问题域的认识更为透彻，并能以人们容易理解的方式表述出来，从而使从需求分析到系统设计的转换更加自然。

同时由于系统是根据应用领域中的真实对象建模的，能更有效全面地理解问题模型的各个方面，整个开发工作更为高效。

2）易于重用

面向对象在分析问题时，要求透过事物表象抓住本质特征，通常在此基础上创建的对象（类）在其问题域中具有普适性，因而可应用于其他类似问题中。

开发人员在创建某个类后，可以在包含该类对象的许多系统中重用它。若新系统具有新特征，可将类扩充，即在保持原有类的所有特性的基础上，通过添加新特征来重用原来的工作成果。

3）便于维护

面向过程的方法强调功能模块（过程）的实现方法，在具体实现时由于函数（过程）与数据的分离，因而软件的开发与维护都较为困难。

而面向对象方法立足于对问题域中的事物（对象）及其相互关系进行透彻分析，因而，在此基础上完成的设计通常比较简洁且易于理解。同时由于将数据和处理这些数据的操作作为一个整体——对象来处理，使得对象相对独立，因而一个对象的修改对其他对象的影响很小，系统开发出的类和对象会比较健壮，从而增强了系统的灵活性和扩展性。

8.1.2 面向对象中的基本概念

1. 对象

从人们的认知角度看，现实世界中的对象是某种确实存在的、可以被感官察觉到的物质，或者是思想、感觉等某种精神的东西。

在面向对象的方法中，人们进行研究的任何事物统称为对象，它可以是有形的实体，如食物、汽车等，也可以表示人的活动，如教学、生产等，也可以表示事件和规则，如战争、法规等。

2. 属性和方法

每个对象都有其独特的性质、状态及行为等特性，在面向对象的方法中，描述对象有两个要素：属性和方法。

属性是描写对象静态特性的数据元素，例如：描述一个人可以采用姓名、性别、身份证号等属性；方法是用于描写对象动态特性（行为特性）的一组操作，例如：每个人都具有工作、学习等行为特性。

3. 类

人类在认知过程中，为了更好地把握客观事物的实质，采用归类方法，即忽略事物的非本质特征，只关心与所研究目标相关的本质特征，并将具有共同本质特性的事物分为同一类。例如，对于各种各样的汽车，你今天看见一辆奔驰（它是一个对象），明天又看见一辆宝

马(它也是一个对象),……,如此这般,假如现在你对汽车产生了兴趣,那么可将这一类对象抽取出它们的共同特征,由此构造一个类——Car 类。

在面向对象的方法中,类是一组具有共同特性的所有对象成员的抽象描述。例如:对上面遇到的汽车进行抽象分析后,可看到汽车都具有汽车型号、车牌、车速、车高、车宽、车身颜色、用油量、前进、后退、加速、减速、制动、转向等特性。

类的定义描述了该类对象共同具有的属性,以及实现该类对象共同行为的具体方法。例如:对于上述 Car 类,可做以下描述(假定对所研究的问题域,下面列出的属性和方法足以解决相关问题)。

类名:Car

属性:

汽车型号、车牌、车速、车高、车宽、车身颜色、用油量等

方法:

前进、后退、加速、减速、制动、转向等

4. 实例化

从一个类定义,可以创建该类的多个"真正实体",即实例,实例是类所定义的对象的具体实现。

实例化是指在类定义的基础上构造对象(实例)的过程。

例如,定义了上述 Car 类后,可以根据该类创建一个个具体的汽车对象(例如一辆牌号为"沪 A·69678"的黑色宝马、另一辆牌号为"京 B·86885"的银色奥迪等)。

5. 继承

继承表达了一种类之间的相互关系,它使得新类可以从已存在的类那里获得已有特征。已存在的类称为"基类"或"父类"(例如"动物"),新建类称为"派生类"或"子类"(例如"狗"或"猫"等)。

继承是面向对象思想很重要的特点。继承带来了诸多好处。

1) 代码复用

将已经存在的类作为基类,子类只需通过继承,就可直接使用基类的程序代码而不必重复劳动,即可以在原有工作成果上,快速开发出高质量的程序。

2) 统一共同属性和行为

继承可以确保基类下的子类都具有基类的属性和行为。若没有继承机制,分别创建的应用于相同问题域中的众多类很有可出现冗余现象和兼容问题。

3) 程序更简洁、更易扩展

继承是一种联结类与类的层次模型,体现了自然界中特殊与一般的关系。在软件开发过程中,继承保证了软件模块的可重用性和独立性,可缩短开发周期,提高软件开发效率,同时使软件易于扩展和维护。

8.2 Python 中的类和对象

Python 是面向对象的程序设计语言。下面简要介绍 Python 面向对象编程的基本方法。

8.2.1 类的定义和对象的创建

1. 类的定义

在 Python 中，类用 class 关键字来定义。

格式为

```
class 类名:                                    #定义一个类对象
    [类变量名 = 初始值]                         #定义类变量(属性)
    [def 方法名(self,参数):                     #定义类方法
        方法体]
```

说明：

(1) 关键字 class 后接的是一个类名，随后是定义类的类体代码，通常由各种各样的定义和声明组成。

(2) 类属性用类中的数据成员(变量)来描述。

(3) 类中的方法类似函数，但类的方法定义中的第一个参数是个特别参数(按惯例为 self)，它在所有的方法声明中都存在，该参数代表实例(对象)本身，当用实例(对象)调用方法时，不需要将 self 传递进来，因为系统会自动将其传入。

(4) 具体对象通过类的实例化获得，格式为

```
对象 = 类名([参数])
```

对象创建后，可获取属性，格式为对象.属性名，并可调用方法，格式为对象.方法名([参数])。

【例 8-2-1】 定义一个简单的 Dog 类，并创建一个对象，该对象调用方法 bark()。

```
class Dog:                                      #定义 Dog 类
    def bark(self):                             #定义方法 bark()
    print("汪!汪!汪!")
dog1 = Dog()                                    #创建对象 dog1
dog1.bark()                                     #对象 dog1 调用方法 bark()
```

运行结果：

```
汪!汪!汪!
```

注意：上述类方法 bark()定义中必须含有参数 self。当对象 dog1 创建后，其调用 dog1.bark()方法时不需要给予参数，因为系统自动将对象 dog1 作为 self 传递给方法 bark()，也即此时的 self 代表 dog1 本身。

【例 8-2-2】 修改上题方法 bark()，在该方法体中增加一个对象(实例)属性 name。

```
class Dog:                                      #定义 Dog 类
    def bark(self,xm):                          #定义方法 bark()
        self.name = xm                          #将 xm(姓名)赋值给对象属性 name
        print("汪!汪!汪!我是" + self.name + "!")
dog1 = Dog()                                    #创建对象 dog1
dog1.bark("阿黄")                               #对象 dog1 调用方法 bark()
```

运行结果:

```
汪!汪!汪!我是阿黄!
```

注意: 上述方法 bark()定义中增加了参数 xm,故当对象 dog1 调用方法 dog1.bark("阿黄")时需要给予一个实际参数"阿黄",系统自动将对象 dog1 作为第一参数 self、将"阿黄"作为第二参数 xm 传递给方法 bark()。

思考:

上述例子主要用于说明 Python 中方法参数的处理方式,若在求解实际问题时仍使用该 bark(self,xm)方法,则会出现什么情况?你认为比较妥当的处理方式是什么?

2. __init__()方法

分析例 8-2-2 中的 bark(self,xm)方法,发现调用该方法时都要给予 xm(姓名)参数,而在真实情况中狗不会只叫一次,即在实际使用时可能要多次调用 bark(self,xm)方法。一般情况下狗的名字起好后不太会变动,故每次调用此 bark(self,xm)方法都给予 xm 的做法显得不太合适。可以在狗对象生成时给予其名字,则以后在需要时直接使用即可(假定在所讨论的问题域中狗名字保持不变)。因而一般在类定义中,可考虑在对象生成的同时对其基本属性赋予具体数值,也即对象的初始化。

在 Python 类中,常常使用__init__()方法(该方法名前后都带有双下划线,在 Python 中,带双下划线的名字具有特殊意义)用于对象的初始化,__init__()方法在类被实例化时立即执行。

【例 8-2-3】 __init__()方法示例。

```
class Dog:                                   # 定义 Dog 类
    def __init__(self,name,color):           # 定义__init__()方法
        self.name = name                     # 为对象的 name 属性赋值
        self.color = color                   # 为对象的 color 属性赋值
    def bark(self):                          # 定义 bark()方法
        print("汪!汪!汪!我是" + self.name + "!")
dog1 = Dog("阿黄","黄色")                      # 创建对象时自动调用__init__()方法
print("刚才创建了一个狗对象,该条狗名叫: " + dog1.name + "颜色为: " + dog1.color)
dog1.bark()
```

运行结果:

```
刚才创建了一个狗对象,该条狗名叫: 阿黄 颜色为: 黄色
汪!汪!汪!我是阿黄!
```

在上述 Dog 类定义中加入了__init__()方法,在该方法中创建了两个对象属性 name 和 color。需要注意的是,执行语句 dog1=Dog("阿黄","黄色"),即创建对象 dog1 时,系统自动调用__init__(self,name,color)方法,即该方法无需显式调用,这就是__init__()方法的特殊之处。由于第一个参数 self 无需处理,故只需传递两个实际参数“阿黄”和“黄色”给予对应的第二个参数 name 和第三个参数 color 即可。

对象的属性可以在类内部使用,如上述第六条语句(类 Dog 内部 print 语句),也可以在类外部使用,如上述第八条语句(类 Dog 外部 print 语句)。

***3. 类变量和对象(实例)变量**

在 Python 面向对象语言中,有两种变量类型——类变量和对象(实例)变量,它们根据

所有者是类还是对象而区分开来。类对象是共享的——它们可以被一个类的所有实例使用，即一个类的变量一经定义后，在其生存期间，任何对象对其所做的修改都会保存并反映到所有对象上。而对象（实例）变量只被自己的对象所拥有，不同对象中的同名对象变量没有任何关联。

为便于理解，下面举例说明：

【例 8-2-4】 在下面的 Dog 类中，添加一个类变量，用于记录狗的数量。

```
class Dog:                                          #定义 Dog 类
    number = 0                                      #定义类变量 number
    def __init__(self,name):                        #定义__init__()方法
        self.name = name                            #为对象的 name 属性赋值
        Dog.number = Dog.number + 1                 #类变量 number 累加
    def bark(self):                                 #定义 bark()方法
        print("汪!汪!汪!我是" + self.name + "!")    #使用实例变量 name
        print("现在有 %d 条狗!" % Dog.number)       #使用类变量 number
dog1 = Dog("阿黄")
dog1.bark()
dog2 = Dog("阿美")
dog2.bark()
dog1.bark()
```

运行结果：

```
汪!汪!汪!我是阿黄!
现在有 1 条狗!
汪!汪!汪!我是阿美!
现在有 2 条狗!
汪!汪!汪!我是阿黄!
现在有 2 条狗!
```

对象 dog1 的实例变量 name 为 dog1 自己所有，经初始化后其 name 值为“阿黄”，对象 dog2 的实例变量 name 为 dog2 自己所有，经初始化后其 name 值为“阿美”，两者不相关。类变量 number 为 dog1 和 dog2 所共享，dog1 和 dog2 生成后，类变量 number 为 2，此时无论是 dog1 还是 dog2，使用该 number 时其值当然为 2。

注意： *在使用类变量时，类变量名前应指明类名，如上例的 Dog.number。*

8.2.2 类的继承

面向对象技术强调软件的可重用性。Python 语言提供了类的继承机制，用于解决软件重用问题。

假设有个 Dog 类完全符合某系统的要求，并已在相关应用场合工作正常，该类如下：

```
class Dog:
    def setName(self,name):
        self.name = name
    def setColor(self,color):
        self.color = color
    def bark(self):
        print("汪!汪!汪!我是" + self.name + "!")
```

现情况发生了变化,有一种特殊的狗——导盲狗加入了系统。显然,一般狗成不了导盲狗,只有经过严格训练,并经过层层筛选能胜任导盲工作的狗才能成为导盲狗。

因而,需要建立一个GuideDog类。那么该GuideDog类是否需要将其所有的类方法和属性完整地重新定义一遍呢?考虑到系统的可扩展性,若每当系统增加新类,每个新类都要完整地重新定义的话,那么整个系统的代码将会变得极其复杂和庞大,并且极可能出现不相容的情况。在面向对象方法中,可通过继承来解决这个问题。

继承是一种强大的功能,通过声明子类(派生类)和已经建立的父类(基类)之间的上下层关系来建立新类,子类继承父类的特性,并加入自己的新特性。在实际使用中可通过继承来实现代码的重用。事实上,大多数的类都是从其他类继承过来的,并根据需要增添自己特有的类方法和属性。

在Python中,类的基类(父类)只是简单地列在子类(派生类)名后面的小括号里。Python支持多重继承,在子类名后面的小括号内,可列出多个基类名,基类名间以逗号分隔。

格式为:

```
class 子类名(基类名1,基类名2…):
    定义子类新特性
```

【例8-2-5】 利用上述的Dog类,定义一个GuideDog类,并创建一个导盲犬对象,该对象调用相关特性。

```
#导入上述已经定义的Dog类
class Dog:
    def setName(self,name):
        self.name = name
    def setColor(self,color):
        self.color = color
    def bark(self):
    print ("汪!汪!汪!我是" + self.name + "!")

#定义GuideDog类
class GuideDog(Dog):                    #继承基类Dog类
    #定义子类自己的__init__()方法
    def__init__(self,name):
        Dog.setName(self,name)          #调用基类的setName()方法
    #定义子类自己的guide()方法
    def guide(self):
        print("我正在引导我的主人!")
#创建一导盲狗对象gDog1
gDog1 = GuideDog("忠诚卫士")
gDog1.bark()                            #调用继承的bark()方法
gDog1.guide()                           #调用自己的guide()方法
```

运行结果:

```
汪!汪!汪!我是忠诚卫士!
我正在引导我的主人!
```

注意：在上述示例中，基类 Dog 类定义中没有采用__init__()方法，而是用 setName()和 setColor()方法为属性 name 和 color 赋值，实际应用时根据情况灵活运用。

在 Python 中，若基类定义中使用__init__()方法，则子类继承分两种情况。

(1) 若子类的初始化与基类完全相同，则子类无须重复定义__init__()方法。在创建子类的实例时，系统会自动调用基类的__init__()方法(参见例 8-2-6)。

(2) 若子类初始化时打算在基类初始化的基础上增添新特性，则子类需要定义自己的__init__()方法。此时，在子类__init__()的方法中应该显式调用基类的__init__()方法(参见例 8-2-7)。

【例 8-2-6】 子类初始化与基类相同示例。

```
#定义基类 Dog 类
class Dog:
    #使用__init__()方法
    def __init__(self,name):
        self.name = name
    def bark(self):
        print("汪!汪!汪!我是" + self.name + "!")
#定义 GuideDog 类
class GuideDog(Dog):
    #定义子类自己的 guide()方法
    def guide(self):
        print("我正在引导我的主人!")
#创建一导盲狗对象 gDog1
gDog1 = GuideDog("忠诚卫士")
gDog1.bark()
gDog1.guide()
```

运行结果：

```
汪!汪!汪!我是忠诚卫士!
我正在引导我的主人!
```

在上例中，子类 GuideDog 的初始化与基类的__init__()方法相同，故无须重新定义，直接使用即可。

【例 8-2-7】 子类初始化在基类初始化基础上增加新特性示例。

```
#定义基类 Dog 类
class Dog:
    #使用__init__()方法
    def __init__(self,name):
        self.name = name
    def bark(self):
        print("汪!汪!汪!我是" + self.name + "!")
#定义 GuideDog 类
class GuideDog(Dog):
    #定义自己的__init__()方法
    def __init__(self,name,year):
```

```
            self.workyear = year                      #增加新属性 workyear
            Dog.__init__(self,name)                   #显式调用基类的__init__()方法
        #定义子类自己的 guide()方法
        def guide(self):
            print("我正在引导我的主人!")
            print("我有%d 年的工作经历!" % self.workyear)
#创建一导盲狗对象 gDog1
gDog1 = GuideDog("忠诚卫士",3)
gDog1.bark()
gDog1.guide()
```

运行结果:

```
汪!汪!汪!我是忠诚卫士!
我正在引导我的主人!
我有 3 年的工作经历!
```

在上例中,子类 GuideDog 在基类 Dog 的初始化基础上增加了新属性(workyear 属性),故在定义自己的__init__()方法时,需使用语句:Dog.__init__(self,name)显式调用基类的__init__()方法。

8.3 面向对象思想应用——图形界面编程

8.3.1 图形用户界面

1. GUI 简介

本章之前所讨论的程序是基于命令行界面的,但在实际应用中,许多应用程序采用的是图形用户界面。

图形用户界面(Graphical User Interface,GUI)又称图形用户接口,是指采用图形方式显示的操作计算机的用户界面。与早期计算机使用的命令行界面相比,图形界面对于用户来说在视觉上更易于接受。

GUI 采用面向用户的设计,其目的是使操作更人性化,减轻使用者的认知负担,使其更适合用户的操作需求。

GUI 的组成部分包括:视窗、图标、菜单、标签和按钮等。

2. GUI 程序开发简介

开发 GUI 程序时,首先需创建一个顶层根窗口,该窗口包含一些小窗口对象,它们共同组成一个完整的 GUI 程序。这些小窗口对象可以是标签、按钮、菜单等,这些独立的 GUI 小窗口对象就是所谓的控件。

通常,用户对控件会有一些操作,例如按下/释放按钮、移动鼠标、在文本框中输入文本、按下 Enter 键等。这些用户行为称为事件,GUI 程序正是由这些伴随其始终的事件体系(除了用户事件外还有系统事件等,本文仅介绍简单的用户事件)所驱动,而 GUI 程序对事件所采取的响应处理称为回调。

一个 GUI 程序启动时,必须先执行一些初始化例程为核心功能的运行做准备。当所有窗口控件(包括顶层窗口)显示在屏幕时,GUI 程序就会进入一个无限事件循环中(等待

GUI 事件、处理事件、再返回等待模式等待下一个事件)。当用户关闭窗口时,必须唤起一个回调来结束程序。

8.3.2 Python 图形框架

1. 简介

Python 图形框架可视为图形领域内的一组基类与应用类之间的交互模式。

Python 的默认 GUI 工具集是 Tk。Tk 最初是为工具命令语言(Tcl)设计的,其流行后被许多脚本语言(包括 Python)采用。Tk 是一个可移植的可以管理窗口、按钮、菜单等的 GUI 工具集,Python 借助 Tk 可快速高效地创建实用程序。

Tkinter 是一个调用 Tcl/Tk 的接口(Tkinter=Tk+interface,正是"Tk 接口"之意),它是一个跨平台的脚本图形界面接口。Tkinter 不是唯一的 Python 图形编程接口,也不是功能最强的一个,但其简单易用且能满足一般 GUI 开发需求,故下面简要介绍 Tkinter 图形编程基础知识。

Tkinter 类支持 15 个核心的窗口控件类,包括 Button(按钮)类、Canvas(画布)类、Menu(菜单)类、Menubutton(菜单按钮)类、Label(标签)类、Message(消息框)类、Listbox(列表)类、Entry (单行输入框)类、Text(多行文本输入框)类、Frame(框架)类、Radiobutton(单选按钮)类、CheckButton(复选按钮)类、Scrollbar(滚动条)类、Scale(标尺进度条)类和TopLevel(顶级窗口容器)类等。

所有这些窗口控件提供了布局管理方法、配置管理方法和控件自己定义的方法。此外,Toplevel 类也提供窗口管理接口。这意味一个典型的窗口控件类提供了大约 150 种方法。

2. Tkinter 创建 GUI 程序

Tkinter 创建 GUI 程序的基本步骤如下:

(1) 导入 tkinter 模块,语句为

```
from tkinter import *
```

或者

```
import tkinter
```

tkinter 模块导入后,就可以使用 tkinter 的类和方法,语句为

```
tkinter.方法名()
```

(2) 创建根窗口(顶层窗口)对象以容纳整个 GUI 中的对象。

根窗口对象由 tkinter 中的 Tk 类创建,语句为

```
root = Tk()
```

若采用 import tkinter 导入 tkinter 的话,则语句为

```
root = tkinter.Tk()
```

(3) 在根窗口对象上创建窗口"控件"对象。

根据需要,在上述所建的根窗口上设置"控件"。通常这些控件会有一些相应的行为,如鼠标单击,键盘按键等,这些称为事件,而程序会根据这些事件采取相应的反应,称为回调。

这个过程称为事件驱动。

(4) 将窗口“控件”对象与对应事件处理程序代码相关联。

(5) 进入窗口事件主循环,语句为

```
root.mainloop( )
```

进入事件循环,接收来自用户的操作(事件),执行相应的事件处理,直到用户关闭窗口。

【例 8-3-1】 创建一个空窗口。

```
from tkinter import *                     # 导入模块
root = Tk()                               # 创建主窗口
root.mainloop( )                          # 进入主循环
```

程序运行后,生成一个空窗口,当用户单击关闭按钮后,窗口才结束运行。

利用 Tk 类创建的窗口,其默认大小为 200×200(像素),默认窗口标题为 Tk。

若要指定窗口大小,可使用 Tk 类的 geometry(参数)方法(参数为字符型,格式为“宽度×高度+左上角水平坐标+左上角垂直坐标”)。

若要设置窗口标题,可使用 Tk 类的 title(参数)方法(参数为字符型,内容为“新窗口标题”)。

【例 8-3-2】 指定窗口标题、大小和初始位置。

```
from tkinter import *
root = Tk()
root.title("我的窗口")                    # 设置窗口标题
root.geometry("500x300 + 0 + 0")          # 设置窗口大小和初始位置
root.mainloop()
```

3. Tkinter 事件处理

一个 Tkinter 应用程序大部分时间处于监听并处理事件的主循环(通过 mainloop()方法进入事件主循环)中。事件包括用户键盘和鼠标操作,以及系统事件(如窗口管理器的重绘事件等)。

Tkinter 提供了强大的事件处理机制。对于任一 GUI 控件对象,可以为事件绑定 Python 处理函数,语句为

```
控件对象.bind(event, handler)
```

上述 event 表示事件,handler 表示对应的处理函数,即回调函数。当发生在窗口控件对象上的事件与所描述的事件匹配时,程序将调用 handler 所指向的回调函数来进行相应处理。

Tkinter 事件都用带左右尖括号的特定字符串描述,下面为常用的鼠标事件:

<Button-1>:鼠标左击事件。

<Button-2>:鼠标中击事件。

<Button-3>:鼠标右击事件。

<Double-Button-1>:鼠标双击事件。

【例 8-3-3】 创建一个空白窗体,当鼠标左击或右击窗体时,显示对应的鼠标事件类型以及所在位置。

```
# 测试鼠标事件
from tkinter import *
root = Tk()
# 定义鼠标事件对应的回调函数 printCoords
def printCoords(event):
    if event.num == 1:
        mouse_click_type = "鼠标左击"
    elif event.num == 3:
        mouse_click_type = "鼠标右击"
    else:
        mouse_click_type = "其他鼠标事件"
    print(mouse_click_type,event.x,event.y)
# 将窗体与鼠标左击事件绑定
root.bind('<Button-1>',printCoords)
# 将窗体与鼠标右击事件绑定
root.bind('<Button-3>',printCoords)
root.mainloop()
```

上述回调函数 printCoords()中的参数 event 为系统传入的事件对象，event.x，event.y 表示当前单击位置的坐标值。

程序运行后，生成一空窗体，当鼠标左击或右击窗体任一位置时，在 Python 的 Shell 窗口内显示对应的鼠标事件和鼠标在该空窗体中单击位置的坐标值。

4. Tkinter 的窗口控件类

1）Label

标签控件用于显示文本或图像，标签可包含多行文本。

利用 Tkinter 的 Label 类可创建标签对象，语句为

```
Label(root, [text = "xxx", width = xx, height = xx])
```

上述第一个参数是所属的父窗口对象，后面为可选参数，其中 text 为欲显示的文本，width 和 height 为标签的宽和高。

若要在应用程序中动态设置标签的显示文本，可使用 Label 的 config 方法，语句为

```
Label 对象.config(None, text = "欲显示的新文本")
```

【例 8-3-4】 创建一个窗口，窗口中包含一标签。

```
from tkinter import *
root = Tk()
#创建 Label 对象 lbl
lbl = Label(root, text = "欢迎使用图形界面", width = 50,height = 10)
#显示 lbl 对象
lbl.pack()
root.mainloop()
```

2）Button

按钮是用于鼠标和按键操作的基本控件。一般按钮对应一个回调函数，当按钮被激活的时候，就调用该回调函数。

利用 Tkinter 的 Button 类可创建按钮对象，语句为

```
Button(root, text = "xxx", command = xx)
```

上述第一个参数是所属的父窗口对象，text 参数为按钮文字，command 参数指明回调函数或命令语句。

【例 8-3-5】 创建一个窗口，窗口中包含一按钮，单击按钮后，在 Python Shell 窗口内显示相应的内容。

```
from tkinter import *
#定义 Button 的回调函数 hello()
def hello():
    print("嗨!你好!")
root = Tk()
#创建 Button 对象 btn,通过 command 属性来指定 Button 的回调函数 hello
btn = Button(root, text = '点我试试!', command = hello)
#显示 btn 对象
btn.pack()
root.mainloop()
```

程序运行后，当单击“点我试试!”按钮后，在 Python 命令窗口中显示“嗨! 你好!”。

思考：

若要使单击按钮后，回调函数的文字显示在应用程序自身窗体内而不是 Python Shell 窗口中，该如何处理？

【提示：

利用 Label 的 config()方法可实现标签文字的动态显示(参见“8.6.2 实验 Python 图形界面编程初步”中的范例 8.6.2-1)。**】**

3) Entry

Entry 是用来收集用户输入的基本控件。

利用 Tkinter 的 Entry 类可创建单行文本框对象，语句为

```
Entry(父窗口,可选属性值)
```

Entry 的常用方法如下：

- insert(index, text)

向文本框中插入值，index：插入位置，text：插入值。

- get()

获取文件框的值。

- delete(index1,[index2])

删除文本框内从位置 index1 至位置 index2(不包括位置 index2)之间的字符，若参数 index2 省略，则只删除位置 index1 上的字符。

【例 8-3-6】 创建一个窗口，窗口中包含上下两个单行文本框，两个文本框中间有一按钮，单击此按钮，可将上文本框中的内容显示于下文本框中。

```
from tkinter import *
root = Tk()
#定义 Button 的回调函数 show()
def show():
        inputTxt = entry1.get()
        entry2.delete(0,END)      # 删除文本框内的所有字符
```

```
        entry2.insert(0,inputTxt)
#创建上文本框对象 entry1
entry1 = Entry(root,width = 30)
entry1.pack()
#创建中间按钮对象 btn
btn = Button(root,text = '显示',command = show)
btn.pack()
#创建下文本框对象 entry2
entry2 = Entry(root,width = 30)
entry2.pack()
root.mainloop()
```

4) Canvas

Canvas(画布)是多用途控件,可用来画图,还可以在画布范围内放置任何控件,语句为

```
Canvas(root, width = 宽度值, height = 高度值)
```

- 画直线。

```
create_line(coords, ** options)
```

其中 coords 表示坐标,例如(1,1,10,10)表示的是画从点(1,1)到(10,10)两点的直线,options 是个变长列表可选参数,可以设定 fill(线条颜色),dash(点样式)和 width(线条宽度)等。

- 画矩形。

```
create_rectangle(bbox, ** options)
```

bbox 表示矩形边界,一般设定矩形的左上角和右下角坐标即可,option 参数同上。

- 画圆(椭圆)。

```
create_oval(bbox, ** options)
```

bbox 为圆(椭圆)外接矩形的左上角与右下角两个点的坐标,option 参数同上。

【例 8-3-7】 用画布绘制直线、矩形和圆。

```
from tkinter import *
root = Tk()
cnv = Canvas(root, width = 200, height = 300)
cnv.pack()
# 使用 create_line 方法绘制直线
cnv.create_line(0, 0, 200, 100,width = 4)
cnv.create_line(0, 100, 200, 0, fill = 'red', dash = (4, 4))
# 使用 create_rectangle 方法绘制矩形
cnv.create_rectangle(50, 25, 150, 75, fill = 'blue')
# 使用 create_oval 方法绘制圆
cnv.create_oval(50,150,150,250, width = 1)
root.mainloop()
```

*5) 菜单

利用 Tkinter 的 Menu 类可创建菜单,步骤为

- 创建菜单栏对象:Menu(父窗口)。

• 创建属于上述菜单栏对象的下拉菜单：Menu(所建菜单栏)。
• 向下拉菜单添加菜单项(命令)。

add_command(label="下拉菜单中菜单项文字"，command="命令或回调函数名")

在创建多个菜单项时，可选 add_separator()添加菜单项分割线。

• 将下拉菜单添加到所建菜单栏中，语句为

所建菜单栏.add_cascade(label="菜单栏中菜单项文字"，menu=所建下拉菜单)。

• 显示所创建的菜单栏。

父窗口.config(menu=所建菜单栏)

【例 8-3-8】 创建一个简单菜单。

```
from tkinter import *
root = Tk()
#定义菜单项对应的回调函数
def openCall():
    print('你单击了 Open 选项')
def saveCall():
    print('你单击了 Save 选项')
def aboutCall():
    print('这是 Menu 菜单的简单演示')
#创建菜单栏 menubar
menubar = Menu(root)
#创建下拉菜单 File,然后将其加入顶级的菜单栏 menubar 中
filemenu = Menu(menubar)                              #生成下拉菜单
#在下拉菜单中添加各项菜单项(命令)
filemenu.add_command(label = "Open", command = openCall)
filemenu.add_command(label = "Save", command = saveCall)
filemenu.add_separator()                              #添加菜单项分割线
filemenu.add_command(label = "Exit", command = root.destroy)
menubar.add_cascade(label = "File", menu = filemenu)  #将下拉菜单增加到菜单栏中
#创建另一个下拉菜单 Help
helpmenu = Menu(menubar)
helpmenu.add_command(label = "About", command = aboutCall)
menubar.add_cascade(label = "Help", menu = helpmenu)
#显示菜单
root.config(menu = menubar)
mainloop()
```

5. Tkinter 的布局管理

在 GUI 编程时，每个控件的布局是很烦琐的，不仅要调整自身大小，还要调整和其他控件的相对位置。

1) Tkinter 的几何布局管理器

Tkinter 提供了三个几何布局管理器：pack、grid 和 place。

• pack 管理器

pack 管理器采用块的方式组织配件，在快速生成界面设计中广泛采用，适用于控件简单的布局。pack 根据控件创建的顺序将控件自上而下地添加到父控件中。

Pack 使用很简单，语句为窗口控件对象.pack(option)。

常用的 option(可选参数)有：

① Side。在父控件中的相对位置,可取值为 TOP(默认),BOTTOM,LEFT,RIGHT。

② Expand。窗口大小变化时控件是否随之同比例变化,可取值为“no”或 0(不变),“yes”或 1(变化)。

③ fill。填充方式,可取值为“X”,“Y”,“BOTH”。

• grid 管理器

grid 管理器采用类似表格的结构组织配件,使用起来非常灵活,用其设计对话框和带有滚动条的窗体效果最好。grid 采用行列确定位置,行列交汇处为一个单元格。

• place 管理器

place 管理器可以精确地指定一个控件的位置和大小,但使用较为复杂。

【例 8-3-9】 简单布局。

```
from tkinter import *
root = Tk()
lbl = Label(root,text = 'Hi!')
lbl.pack()
btn1 = Button(root,text = 'BUTTON1')
btn1.pack(side = LEFT)
btn2 = Button(root,text = 'BUTTON2')
btn2.pack(side = RIGHT)
root.mainloop()
```

考虑一下,若将上例中的第六、第八行 pack()中的参数 side 去掉,则上述控件如何显示?若将第四行改为 lbl.pack(side=BOTTOM),则上述控件又如何显示?

2) Frame 控件

Frame(框架)是屏幕上的一块矩形区域,作用类似容器,通过将其他控件置于其中来布局窗体。

语句为 Frame (父窗口)。

若有必要,可在 Frame( )中选用相应参数对框架做进一步的管理。

【例 8-3-10】 简单框架应用。

```
from tkinter import *
root = Tk()
root.geometry('500x70 + 0 + 0')
#创建 Frame 对象 frame1
frame1 = Frame(root)
frame1.pack()
#创建 Frame 对象 frame2
frame2 = Frame(root)
frame2.pack(side = BOTTOM)
#在 frame1 中创建标签对象 lbl1
lbl1 = Label(frame1, text = "用户名",width = 8)
#在 frame1 中创建文本框对象 entry1
entry1 = Entry(frame1)
lbl1.pack(side = LEFT)
entry1.pack(side = RIGHT)
#在 frame2 中创建标签对象 lbl2
```

```
lbl2 = Label(frame2, text = "口令",width = 8)
#在 frame2 中创建文本框对象 entry2
entry2 = Entry(frame2)
lbl2.pack(side = LEFT)
entry2.pack(side = RIGHT)
root.mainloop()
```

8.4 本章小结

本章首先简要介绍了面向对象思想、面向对象的优点和面向对象中的基本概念,然后叙述了 Python 面向对象编程的基本方法,之后,作为面向对象思想的应用,介绍了 Python 图形用户界面编程的初步知识。

本章要点如下:

(1) 面向对象思想强调以现实世界中的事物为中心来分析思考问题,力求把握事物的本质特性以及事物之间的相互作用和相互联系,因而更符合人们的思维方式,可使构建的计算机世界更接近真实世界。

(2) 面向对象方法包含对象、类和继承等要素。在面向对象方法中,人们进行研究的任何事物统称为对象,类是对一组具有共同特性(属性和方法)的对象的抽象,继承则表达了一种对象类之间的层次关系,它使得某类(子类)可以继承另一类(父类)的已有特性。继承性是面向对象方法不同于其他设计方法的重要特点。

(3) Python 是面向对象的程序设计语言,借助 Python 可实现面向对象思想。

(4) GUI 是方便用户操作的图形用户界面,GUI 由视窗、图标、菜单、标签和按钮等窗口对象组成。

(5) Python 默认的 GUI 工具集是 Tk,Tkinter 是一个调用 Tcl/Tk 的接口,其是一个功能简单且能满足一般 GUI 开发需求的跨平台脚本图形界面接口。Tkinter 类支持 Button、Label、Entry、Frame、Canvas 等 15 个核心窗口控件类,所有这些窗口控件提供了布局管理方法、配置管理方法和控件自己定义的方法。

(6) 利用 Python 提供的图形框架类可编写出实用的 GUI 程序。

8.5 习题与思考

8-1 单选题。

(1) 下面选项中,不属于面向对象要素的是________。

A. 对象　　B. 类　　C. 过程　　D. 继承

(2) 下面关于对象属性和方法的叙述中,正确的是________。

A. 属性是描写静态特性的数据元素,方法是描写动态特性的一组操作

B. 属性是描写动态特性的一组操作,方法是描写静态特性的数据元素

C. 属性是描写内在静态特性的数据元素,方法是描写外在静态特性的数据元素

D. 属性是描写自身动态特性的一组操作,方法是描写作用于外界的动态特性的一组操作

(3) 下面关于面向对象方法优点的叙述中,不正确的是________。

A. 符合人类习惯的思维方法　　B. 以功能分析为中心

C. 良好的可重用性　　D. 良好的可维护性

(4) 当 Python 中的一个类定义了________方法时,类实例化时会自动调用该方法。

A. auto()　　B. __auto__()

C. init()　　D. __init__()

(5) 下面关于类继承的叙述中,错误的是________。

A. 一个基类可以有多个子类,一个子类可以有多个基类

B. 继承描述类的层次关系,子类可以具有与基类相同的属性和方法

C. 一个子类可以作为其子类的基类

D. 子类继承了父类的特性,故子类不是新类

(6) 为使 Tkinter 创建的 GUI 程序进入窗口事件主循环,顶层窗口对象应调用________方法。

A. wait()　　B. enter()　　C. mainloop()　　D. start()

(7) 下面 Tkinter 的控件类中,________可用于创建单行文本框。

A. Button　　B. Label　　C. Entry　　D. Text

(8) 为使 Tkinter 创建的按钮能起作用,应在创建按钮时,利用按钮控件类的________参数指明回调函数或命令语句。

A. root　　B. command　　C. text　　D. bind

(9) 下面选项中,________可用于将 Tkinter 创建的控件放置于窗体。

A. pack　　B. show　　C. set　　D. bind

(10) 下面 Tkinter 的控件类中,________可用于画矩形、椭圆等图形。

A. Frame　　B. Message　　C. Menu　　D. Canvas

8-2　思考题。

(1) 对象方法和一般函数有何区别?

(2) 继承的含义是什么?

(3) Python 中的类变量与对象(实例)变量有何区别? 类变量有何作用?

(4) Tkinter 创建一般 GUI 应用程序有哪些基本步骤?

(5) 如何理解事件驱动机制?

8.6 实　验

8.6.1 Python 面向对象编程初步

1. 实验目标

(1) 了解 Python 语言面向对象编程的基本方法。

(2) 掌握简单类的定义和对象操作的方法。

(3) 理解__init__( )函数的含义、作用与执行过程。

(4) 理解类继承的概念,能够定义和使用类的继承关系。

2. 实验范例

【范例 8.6.1-1】 类的定义和对象的创建。

定义一个矩形类,要求其有计算周长、面积以及矩形图形显示等方法(功能),并依据该类创建对象进行简单测试。

(1) 分析。

此处讨论的矩形为非倾斜矩形,即矩形四边都是水平或垂直方向,因而只要确定其左上角和右下角的 x、y 坐标即可,故该矩形类包含 4 个数据成员:left,top(左上角坐标),right,bottom(右下角坐标),考虑采用__init__( )函数对数据成员赋值,用方法 getPerimeter(),getArea(),draw()分别实现计算周长、面积和矩形显示等功能。

(2) 程序实现。

```
#定义 Rectangle 类
class Rectangle:
    def __init__(self,x1,y1,x2,y2):
        self.left = x1
        self.top = y1
        self.right = x2
        self.bottom = y2
        #获取矩形的宽和高
        self.width = self.right - self.left
        self.height = self.bottom - self.top
    def getPerimeter(self):
        perimeter = 2 * (self.width  + self.height)
        return perimeter
    def getArea(self):
        area = self.width * self.height
        return area
    def draw(self):
        print("左上角:(%d,%d),右下角:(%d,%d)"%(self.left,self.top,self.right,self.
bottom))
        print ("[此处画出本矩形!]")
#创建对象并测试
rec1 = Rectangle(2,3,12,8)
print("rec1 的图形是:")
rec1.draw()
print("rec1 的周长是:",rec1.getPerimeter())
print("rec1 的面积是:",rec1.getArea())
```

(3) 思考。

- 上述的 Rectangle 类采用__init__( )函数对数据成员赋值,若不采用该函数,那如何获取矩形的左上角和右下角坐标?
- 上述的 draw( )方法只是做了象征性的处理,若想显示出矩形图形,可尝试采用 print 语句“画出”矩形的大致图形。

修改上述类中的 draw( )如下:

```
def draw(self):
```

```
    #此处仅画出形状,不考虑坐标
    print("#" * self.width)
    for i in range(0,self.height - 2):
        print("#" + " " * (self.width - 2) + "#")
    print("#" * self.width)
```

然后创建矩形对象测试之。

上述修改过的 draw()也只是粗略地“勾画”出矩形外貌,有兴趣的同学可以在学习了 Python 图形界面编程后,用图形类画出标准的矩形。

【范例 8.6.1-2】 类的继承。

某高校欲用现代信息技术管理教学事务。作为练习,现要求创建教师和学生相关的类,并作简单测试。

(1) 分析。

教师和学生有很多共同的特征,如姓名、性别和出生日期等,此外他们还拥有一些不同的特征,如教师有职称、薪水、教学任务等,学生有学费、所学课程成绩等。虽然可以分别为师生创建两个独立的类进行处理,但每增加一个共同的特征就意味着在两个类中都要同时增加该特征,考虑学校还有其他工作人员,人员之间都具有相同特征,若都分开创建独立类的话,将会使系统臃肿不堪,并且极可能产生不相容的情况。一个好的方法是创建一个基类 Member,然后让教师和学生类分别继承它成为 Member 的子类,当系统扩展加入许多其他类型的人员时,只要使新人员类继承 Member 基类,并添加自己特有的属性和方法即可。如此,可大大提高类的处理效率。

(2) 程序实现。

下面是一小段示例代码:

```
#定义基(父)类
class Member:
    def setInfo(self, xm,xb,lb):
        self.name = xm
        self.gender = xb
        self.type = lb                  #type 表示人员类别(lb)
    def show(self):
        print(self.name,self.gender,self.type)
#定义教师类
class Teacher(Member):
    def __init__(self, xm, xb, lb):
        Member.setInfo(self, xm,xb,lb)
        self.lecture = []               #所教课程
    #方法 setLecture()输入教师所教的课程
    def setLecture(self):
        Lectures = input("请输入" + self.name + "所教课程(空格分隔,按 Enter 键结束)")
        for t in Lectures.split():
            self.lecture.append(t)
    def show(self):
        Member.show(self)
        print("所教课程有:",self.lecture)
#定义学生类
```

```
class Student(Member):
    def __init__(self, xm, xb, lb):
        Member.setInfo(self, xm,xb,lb)
        self.course=[]                    #所学课程
        self.score=[]                     #所学课程对应的成绩
    #方法 setScore ()输入学生所学课程和成绩
    def setScore(self):
        Courses=input("请输入"+self.name+"所学课程(空格分隔,按 Enter 键结束)")
        for t in Courses.split():
            self.course.append(t)
        Scores=input("请输入所学课程对应的成绩(空格分隔,按 Enter 键结束)")
        for t in Scores.split():
            self.score.append(int(t))
    def show(self):
        Member.show(self)
        print("所学课程为:",self.course,"对应成绩为",self.score)
#创建对象并测试
t1=Teacher("张明","男","教师")
t1.setLecture()
t1.show()
s1=Student("李丽","女","学生")
s1.setScore()
s1.show()
```

对所建类进行测试时,可先创建一教师对象 t1,同时用基本信息(如"张明,男,教师")初始化为该对象,然后根据该对象的授课情况输入具体信息(数据自拟),接着测试 t1 对象显示的信息是否正确;同理,创建学生对象 s1,同时用基本信息(如"李丽,女,学生")初始化为 s1 对象,然后根据 s1 对象的具体情况,输入 s1 所学课程和对应成绩(数据自拟),接着测试 s1 对象显示的信息是否正确。

3. 实验内容

(1) 定义一个简单类:点(Point)类,点的位置由屏幕水平坐标 x,垂直坐标 y 表示,要求用合适的方法初始化点的起始位置,然后定义一个方法 move()实现点的移动,再定义一个方法 show()显示当前点的坐标。创建一个对象验证之。

请在下面程序的空白处填入代码,完成题目要求。

```
#定义 Point 类
class Point:
    def __init__(self,x1,y1):   #初始化点的起始位置
        self.x=________(1)________
        self.y=________(2)________
    def move(self,newX,newY):         #移动到新位置(newX,newY)
        ________(3)________ =newX
        ________(4)________ =newY
    def show(self):
        print ("现在点的位置为:(%d,%d)"%(self.x,self.y))
#创建对象并测试
p1=________(5)________                #创建一 Point 对象 p1,起始位置为(2,3)
p1.show()
________(6)________                   #对象 p1 移动到(5,6)
```

```
print("点移动后情况")
p1.show()
```

(2) 创建一个简单的汽车(Car)类,用变量 id 和 curSpeed 分别表示车牌号和当前车速,用方法 changeSpeed()表示改变汽车的速度,用方法 stop()表示停车。创建一个汽车对象,并作简单测试。

请在下面程序的空白处填入代码,完成题目要求。

```
#定义 Car 类
class Car:
    def __init__(self,num,speed = 0):
        self.id = num
        self.curSpeed = speed
    #getID()方法获取车牌号
def getID(self):
        return ________(1)________
    #getCurSpeed()方法获取当前车速
def getCurSpeed(self):
        return ________(2)________
    def changeSpeed(________(3)________, ________(4)________):
        self.curSpeed = newSpeed
    def stop(self):
        self.curSpeed = 0

#创建对象并测试
c1 = Car("沪 A1567")
print("车牌号为" + c1.getID() + "的车起始车速是:",c1.getCurSpeed())
c1.changeSpeed(80)
print(c1.getID() + "变速后,当前车速是:", ________(5)________)
c1.stop()
print(c1.getID() + "停车后,当前车速是:",c1.getCurSpeed())
c2 = Car("沪 B6567",20)
print("车牌号为" + c2.getID() + "的车起始车速是:",c1.getCurSpeed())
```

(3) 设计一个可实现基本银行存储业务的账户(Account)类,包括的变量有"账号(id)"和"账户余额(balance)",包括的方法有"存款(deposit)"、"取款(withdraw)"和"显示余额(display)"。

要求定义 Account 类,创建账户类的对象,完成对象的初始化(赋予账号和初始存款),并利用该对象进行存款、取款和显示账户余额等操作。

(4) 假设某一小公司打算采用现代信息手段进行人员管理。该公司现有人员为:经理(manager)、技术人员(technician)和销售人员(saleman)。试用类的继承和相关机制实现下述功能:

① 人员基本信息的显示;

② 计算并显示员工月薪(月薪计算办法:经理拿保底月薪 8000 元,技术人员按每小时 30 元领取月薪;销售人员的月薪按当月销售额的 5%提成)。

【提示:

设计一个基类:员工类(Employee),用来描述所有员工的共同特性,该类应有姓名、性别、职位、薪水等基本信息,并应提供一个显示员工基本信息的方法 showInfo()。经理、技

术人员和销售人员对应的类都继承 Employee 类,并添加各自的特性(经理增加"保底月薪"属性,技术人员增加"月工作时间"属性,销售人员增加"月销售额"属性),根据月薪计算办法,编程实现各自月薪的处理方法 getSalary()。】

*(5) 定义出租汽车 Taxi 类,并创建一对象 tx1,该对象根据乘客所乘的车程收取出租费。编写代码实现。

【提示:

Taxi 类可从上述第(2)题 Car 类继承而来成为 Car 类的子类,然后添加 Taxi 类自身方法:setUnitPrice()(定义每公里单价)和 charge()(收取出租费)。setPrice()接收传入的参数 dj(单价)并将其赋值给对象属性 unitprice(出租费单价),charge()接收乘客所乘的车程参数 distance,经处理后(处理方式自拟),要求乘客付费。为配合模拟,在 Taxi 类中添加 getOn()方法,该方法内容为 print("乘客上车!")。

模拟场景:对象 tx1 载乘客上车,车(tx1)启动前进,过了一段时间到达目的地,(tx1)刹车,然后 tx1 调用 charge()方法,收取乘客的出租费。】

8.6.2 Python 图形界面编程初步

1. 实验目标

(1) 了解 Python 图形界面编程的基本方法。

(2) 进一步体会面向对象思想的实际应用。

2. 实验范例

【范例 8.6.2-1】 动态标签。

创建包含一个标签和两个按钮的图形界面程序。单击第一个按钮在标签中显示"你好!",单击第二个按钮在标签中显示"祝你成功!"(界面如图 8-6-1 所示)。

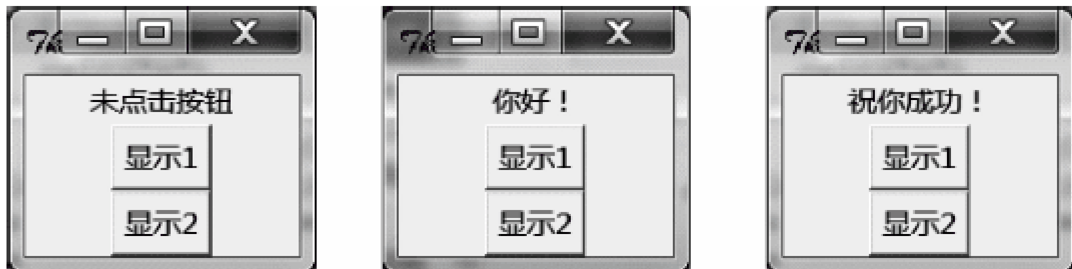


图 8-6-1 范例 8.6.2-1 图

(1) 分析。

要在同一标签中动态显示文本,可利用 Label 的 config 方法实现。

(2) 程序实现。

```
from tkinter import *
root = Tk()
#定义按钮对应的回调函数
def show1():
    lbl1.config(None,text = "你好!")
def show2():
    lbl1.config(None,text = "祝你成功!")
#创建标签和按钮
lbl1 = Label(root, text = '未单击按钮')
lbl1.pack()
```

```
btn1 = Button(root, text = '显示 1', command = show1)
btn1.pack()
btn2 = Button(root, text = '显示 2', command = show2)
btn2.pack()
root.mainloop()
```

【范例 8.6.2-2】 标签和按钮。

创建一图形界面程序，该界面包含一个标签和一个按钮，单击按钮后，标签显示单击的次数(界面如图 8-6-2 所示)。

(1) 分析。

根据题意，按钮对应的回调函数应能统计单击次数，故需使用全局变量来处理，同时为了显示最新的单击次数，需动态刷新标签。

(2) 程序实现。

```
from tkinter import *
root = Tk()
count = 0                                   # count 用于记录单击次数
# 定义按钮的回调函数 show()
def show():
    global count
    count += 1
    lbl1.config(None,text = "按钮被单击了" + str(count) + "次!")   # 动态刷新标签显示
lbl1 = Label(root, text = '未单击按钮')
lbl1.pack()
btn = Button(root, text = '点我试试!', command = show)
btn.pack()
root.mainloop()
```

【范例 8.6.2-3】 文本框、按钮和 Frame(框架)。

编程实现简单的加法运算(界面如图 8-6-3 所示)。该界面包含三个文本框(被加数、加数和结果文本框)、两个标签(加号和等号标签)以及一个按钮(计算按钮)，在被加数和加数文本框输入整数后，单击“计算”按钮，在结果文本框中显示计算结果。

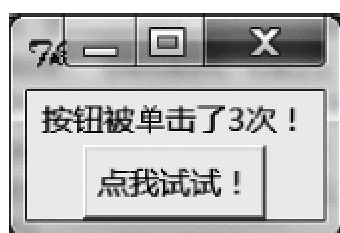


图 8-6-2 范例 8.6.2-2 图



图 8-6-3 范例 8.6.2-3 图

(1) 分析。

为使本题的各个控件能按题目界面要求显示，可使用框架来进行控件的布局。根据题意，需要创建一个框架，以便将三个文本框(两个加数框与结果框)和两个标签(加号和等号标签)放置其中。

(2) 程序实现。

```
from tkinter import *
root = Tk()
# 创建一个 Frame 以便容纳所需的文本框和标签
```

```
frame1 = Frame(root)
frame1.pack()
#在Frame1中创建三个文本框(被加数和加数框与结果框)和两个标签(加号和等号)
entry1 = Entry(frame1,width = 8)
entry2 = Entry(frame1,width = 8)
entry3 = Entry(frame1,width = 8)
lbl1 = Label(frame1, text = " + ",width = 3)
lbl2 = Label(frame1, text = " = ",width = 3)
#将三个文本框和两个标签置入Frame1中
entry1.pack(side = LEFT)
lbl1.pack(side = LEFT)
entry2.pack(side = LEFT)
lbl2.pack(side = LEFT)
entry3.pack(side = RIGHT)
#定义"计算"按钮对应的回调函数calc()
def calc():
    num1 = int(entry1.get())
    num2 = int(entry2.get())
    num3 = num1 + num2
    entry3.delete(0,END)
    entry3.insert(0,num3)
#创建"计算"按钮
btn = Button(root,text = "计算",command = calc)
btn.pack()
root.mainloop()
```

3. 实验内容

(1) 编写一个GUI程序,该程序包含两个文本框和一个按钮,在第一个文本框输入一个整数(1~7)后,单击按钮,在第二个文本框中显示对应的英语星期表示(界面如图8-6-4所示)。

(2) 编程实现一个简单计算器(界面如图8-6-5所示)。单击第二行的+、-、*、/和**按钮时,程序对输入的两个整数进行相应运算,运算结果显示在右边的文本框中,同时将左边的两个文本框之间的标签更新为所单击的运算符号。



图8-6-4 英语星期表示

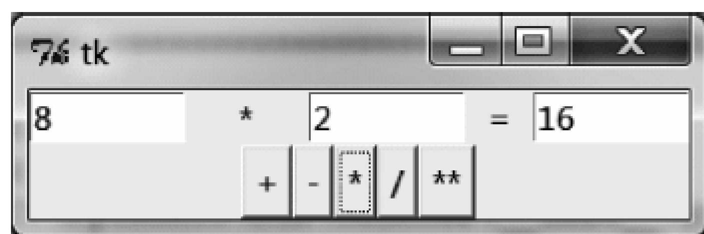


图8-6-5 简单计算器

(3) 编写一个跟踪鼠标位置的图形界面程序,鼠标单击时在所处位置绘制一个十字,同时在窗口上方显示鼠标所在位置的坐标,鼠标双击时擦除所有的十字(界面如图8-6-6所示)。

【提示:

创建一窗口,窗口上部放置一标签,用于显示鼠标所在位置的坐标,标签下面大部分空间放置一画布Canvas,用于在鼠标单击位置画十字。】

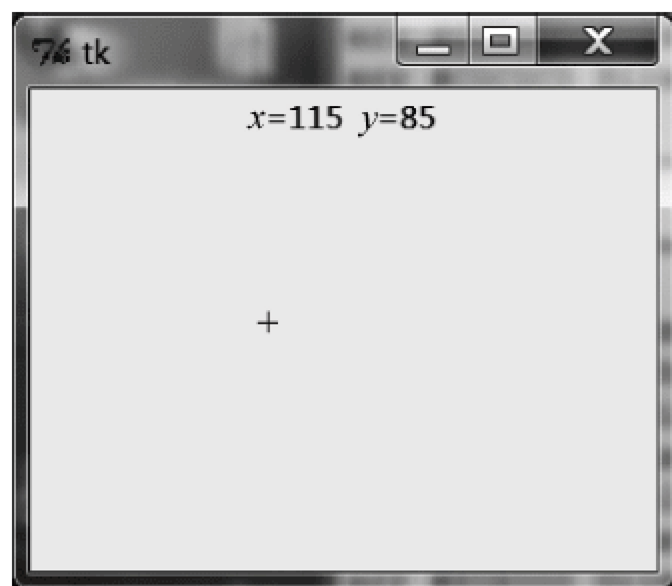


图8-6-6 跟踪鼠标位置

鼠标单击事件对应的回调函数的关键代码如下:

```
……
#显示鼠标所在位置的坐标
标签对象.config(None,text = "x = " + str(event.x) + " y = " + str(event.y))
#画十字
画布对象.create_line(event.x ,event.y - 5 ,event.x ,event.y + 5 , width = 1)
画布对象.create_line(event.x - 5 ,event.y ,event.x + 5 ,event.y , width = 1)
……
```

鼠标双击事件对应的回调函数的关键代码如下：

```
……
标签对象..config(None,text = "")
#以比画布对象略大的矩形(填充色为 WhiteSmoke)擦除画布上的所有十字
画布对象.create_rectangle(0, 0,画布宽度 + 2, 画布高度 + 2, fill = 'WhiteSmoke')
……
```

*(4) 编写程序，使用 Canvas 控件，画出各种直线、圆、矩形等图形。

*(5) 编写程序，在窗口中添加菜单栏，在菜单栏添加菜单项，并添加下拉菜单，通过选择菜单项可以执行不同的操作(菜单栏、菜单项和相应操作自拟)。

附录 A 习题解答

第 1 章

1-1　B

1-2　逻辑错误

1-3　高级语言

1-4　分布式程序设计

1-5　支持面向对象、内存自动回收、支持动态类库和外接函数库、跨平台的开源高级语言。

1-6　C 语言

1-7　网络编程、图形编程、数学应用、数据库应用、多媒体应用、Web 编程。

1-8　www. python-org

1-9　. pyc

1-10　input()

1-11　print("姓名")

第 2 章

2-1　B　D　D　C　A

2-2

① 使变量 i 依次取值 100～999。

② 对于 i 的每一取值,分别取出个位 C、十位 B、百位 A。

③ 若 $i-(C\times10+B)==(C\times10+A)$,输出 A、B、C。

或

① 使变量 a 依次取值 1～9。

② 对于 a 的每一取值,使变量 b 依次取值 0～9。

③ 对于 a、b 的每一取值,使变量 c 依次取值 1～9。

④ 若 $(a\times100+b\times10+c)-(c\times10+b)==(c\times10+a)$,输出 a、b、c。

2-3

① 使变量 i 依次取值 0、1、2、3、4。

② 对于 i 的每一取值,使变量 j 依次取值 0、1、2、3。

③ 若 i、j 不同时为 0,输出邮资值($3\times i+5\times j$)。

2-4

① 第一步为 1,第二步为 1,第三步为 2。

② 循环,从第 4～第 30 步。

③ 每一步的爬法数等于前一步的数＋前三步的数。

④ 输出第 30 步爬法数。

2-5

输入总数 n

$a=0,b=0$

$c=2$

如果 $n>0$，循环：

　　如果$n==1$：

　　　　$a=a+1$

　　　　$n=n-1$

　　跳出循环

　　否则：

　　如果$b==2$ 或 $c==2$：

　　　　$a=a+2$

　　　　$c=0$

　　　　$n=n-2$

　　如果 $n==0$：

　　跳出循环

如果$n==1$：

　　$b=1$

否则：

　　输入 b 数据(1 或 2)

　　如果$b==1$：

　　　　$c=c+1$

　　$n=n-b$

输出("$a=$",a)

第 3 章

3-1　A　B　F　H　I

3-2　B　F　G　I　J

3-3　B　C　D　F　G　I

3-4　$a=12,b=10,c=6,d=1,e=2.5,f=3.5,g=2,h=2.0,i=8.333333333333332$，$j=$False，$k=$True

3-5　(1) 4

(2) hello world

hello world

hello world

(3) Hd (4) Birth (5) Happyday

3-6　(1) L3：[[2，'two']，[3，'three']，[4，'four']]

　　(2) L3[1,1]：'three'

(3) L3:[[2, 'two'], [3, 'three']],L4:[4, 'four']

(4) L3:[[2, 'two'], [3, 'three'], 4, 'four']

3-7 (1) $a-b$ (2) $a-(a-b)$或 $a\&b$ (3) a ^ b (4) len($a|b$)

3-8

正确的:(1)、(5)

错误:(2) 类型构造器中键值是标识符表示,不能加引号。

(3) 集合的字面常量表示中,键与值之间用:间隔,键和值都是数据常量。

(4) 集合的字面常量表示中,键是数据常量,字符串要加引号。

3-9 思考在下面举出的应用环境中适合的数据类型,试在 Python Shell 中举实例表示,并测试它们的主要操作。

(1) 100 以内的素数。

以集合表示,例如建立一个 100 以内素数的集合。

```
>>> primes = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97}
>>> # 对于任意一个 100 以内的整数,可以判断是否是一个素数
>>> x = 28
>>> x in primes
False
```

(2) 1~10 的数字的阶乘。

以元组 F 表示,求任意一个 10 以内的阶乘可以通过查表法完成,例如求 5 的阶乘。

```
>>> F = (0,1,2,6,24,120,720,5040,40320,362880,3628800)
>>> i = 5
>>> F[i]
120
```

元组与集合的不同在于它的存储顺序是确定的,集合没有顺序。用查表法查阅阶乘需要下标来确定所查值,所以不能用集合。而素数表中只需有素数的全集,存储位置不重要。

(3) 斐波那契数列。

以列表表示,从 1,1 出发,可以顺次求解每一位斐波那契数列。

```
>>> L = [1,1]
>>> i = 0
>>> j = 1
>>> L.append(L[i] + L[j])
>>> L
[1, 1, 2]
>>> i = i + 1
>>> j = j + 1
>>> L.append(L[i] + L[j])
>>> L
   [1, 1, 2, 3]
```

(4) 班级考试成绩单。

以嵌套的列表表示,例如期末考试有三门主课,scores 列表中包括 5 位同学的成绩,每位同学一个子表,子表中包括三门课程的成绩。

```
>>> scores = [[78,90,88],[45,67,82],[90,92,95],[78,73,62],[94,85,81]]
>>>#求第一门课程的总分
>>> grades1 = scores[0][0] + scores[1][0] + scores[2][0] + scores[3][0] + scores[4][0]
>>> grades1
385
>>> #增加一名学生的成绩
>>> scores.append([78,85,91])
>>> scores
[[78, 90, 88], [45, 67, 82], [90, 92, 95], [78, 73, 62], [94, 85, 81], [78, 85, 91]]
>>>#显示第二位同学的成绩
>>> scores[1]
[45, 67, 82]
```

(5) 自定义的英汉字典。

```
>>> mydict = dict(ecnu = "华东师范大学",function = "函数",datatype = "数据类型")
>>> mydict[ecnu]
>>> mydict["ecnu"]
'华东师范大学'
>>> mydict.get("program","not found!")
'not found!'
```

第 4 章

4-1　B

4-2　A

4-3　D

4-4

```
for x in range(0,13):
    for y in range(0,13):
        if(x + y == 12 and 2 * x - 3 * y == 14):
            print("x = " + str(x) + ",y = " + str(y))
            break
```

4-5

```
myid = input("输入身份证:")
print("您的出生日期为:")
print(myid[6:10])
print("年")
print(myid[10:12])
print("月")
print(myid[12:14])
print("日")
if ((int)(myid[ - 2]) % 2 == 0):
    print("女")
else:
    print("男")
```

4-6

```
numHeads = 35
numLegs = 94
```

```
for numChickens in range(0, numHeads + 1):
    numRabits = numHeads - numChickens
    if 2 * numChickens + 4 * numRabits == numLegs:
        print("numChickens:",numChickens)
        print("numRabits:",numRabits)
        break
```

4-7

```
linenum = input("请输入行数")
for i in range(1,(int(linenum) + 1)):
    print(" " * (int(linenum) - i),end = '')
    print(" * " * (2 * i - 1))
```

4-8

```
i = 1
li = []
while(i < 11):
    num = input("请输入第" + str(i) + "个数字")

    num = int(num)
    if num in li:
        print("数字有重复")
        continue
    else:
        i = i + 1
    li.append(num)
li.sort()
print(li)
```

第 5 章

5-1 （1）

```
>>> x,y = input("请输入一组坐标值").split(',')
请输入一组坐标值 45,67
>>> x,y
('45', '67')
```

（2）

```
>>> a,b,c = input("请输入三个浮点数：").split()
请输入三个浮点数：24.8 67.8 90.5
>>> a,b,c
('24.8', '67.8', '90.5')
```

空格键和 Tab 键都可。

（3）

```
>>> for i in range(1,5):
    L.append(input())

one
two
```

```
three
four
>>> L
['one', 'two', 'three', 'four']
```

5-2

```
>>> outstr1 = "result: %d-%d-%d %d:%d:%d" % (y,m,d,hh,mm,ss)
>>> outstr1
'result:2014-8-15 13:23:57'
>>> outstr2 = "result:" + repr(y) + '-' + repr(m) + '-' + repr(d) + ' ' + repr(hh) + ':' + repr(mm)
 + ':' + repr(ss)
>>> outstr2
'result:2014-8-15 13:23:57'
```

5-3

参考一：

```
for i in range(2,6):
    print(L[i],end=',')
print(L[i],end='.')
```

参考二：

```
outstr = ''
for i in range(2,6):
    outstr = outstr + repr(L[i]) + ','
outstr = outstr + repr(L[i]) + '.'
print(outstr)
```

5-4

```
for i in range(0,len(L)):
    f.write(repr(L[i]))
    if (i+1) % 5 == 0:
        f.write('\n')
    else:
        f.write(' ')
```

5-5

参考程序：

```
filename = input("file name:")
f = open(filename)
a = f.read()
#处理标点
a = a.replace('.',' ')
a = a.replace(',',' ')
a = a.replace('!',' ')
a = a.replace(';',' ')
a = a.replace('-',' ')
#得到所有的单词表
L = a.split()
for i in range(0,len(L)):
```

```
        L[i] = L[i].lower()

#得到不重复排序的单词表
words = set(L)
words = list(words)
words.sort()
#计算 words 中单词在 L 中出现的次数,得到单词频率列表 wordlist
wordlist = []
for x in words:
        wordlist.append([x,L.count(x)])
f.close()
#显示单词频率列表 wordlist,并写入文件 wordlist.txt
f = open("wordlist.txt","w")
f.write('Word frequency table:\n')
print('Word frequency table:\n')
i = 0
for x in wordlist:
        f.write('% - 15s% - 5d'%(x[0],x[1]))
        print('% - 15s% - 5d'%(x[0],x[1]),end = "")
        i = i + 1
        if i % 3 == 0:
                f.write('\n')
                print()

f.close()
```

5-6　在程序发生语法错误和运行错误(内置异常)时,并且发生该错误的语句没有在 try 语句中或已经被 try 子句捕捉到但没有找到相匹配的 except 子句时,“默认异常处理器”被触发启动。对所发生的异常,“默认异常处理器”的处理手段是:输出相关的出错信息、终止程序的运行。

5-7　一句不带错误名的 except 子句,必须放在所有带有明确错误名的 except 子句后面,有了它,try 子句中的任何错误将不会引发“默认异常处理器”的启动。因此它不适用于程序的调试。但在程序的发布版中,可以用它给出比较笼统的相对友好的错误提示,以免程序时常发生莫名其妙的崩溃。

5-8　尽管 else 子句看似可以完全合并到 try 子句中,但从编程逻辑的角度来看,将 else 子句单独罗列出来,在其中放置无异常发生后所要进行的处理,这样的结构使得逻辑更清晰,更易于理解。

5-9　不管 try 语句中是否有错误发生、所发生的错误是否能被 except 匹配,finally 子句中的语句肯定会在退出 try 语句前被执行,即使 except 子句中有终止程序运行的指令,finally 子句也会在整个程序被终止前得到运行。因此,打扫战场的扫尾工作可以放在 finally 子句中。

5-10　Python 系统内部,会在发生错误时自动引发内置异常,而 raise 语句是让编程者人为地去引发内置异常或用户自定义异常。

第 6 章

6-1　C　A　B　A　D

6-2　函数定义时的参数称为形式参数(简称形参),函数调用时的参数称为实际参数

(简称实参)。形式参数是在函数定义中出现的,一般是变量形式,但未发生函数调用时系统并不为其分配内存空间。实际参数是在函数调用时传给函数的,可以是变量,也可以是常量或表达式。它们是系统实实在在分配了内存空间的。

函数调用时所提供的实参个数应与函数定义时的形参一致,两者位置也要互相对应。但并不要求对应的形参和实参变量名字相同,因为它们是不同作用域中的变量。

6-3 局部变量是指在函数中定义的变量,一个函数所带的参数也是局部变量。局部变量的作用域只是该函数内部,所以不同的函数中可以有相同名称的变量,它们在各自的函数中互不干扰。当函数执行完毕,局部变量所占有的内存空间也被释放。

在所有函数外定义的变量为全局变量。全局变量既可用在主程序中,也可以在各函数中使用。但程序中过多使用全局变量,将使函数间的耦合变得紧密,破坏函数的独立性。

6-4 不允许。在 Python 函数定义中,不允许只有函数定义头部,后面不带任何语句(这与 C 语言系列不同,因为 C 语言系列是用大括号来确定函数体结构的)。如果确实因某种需要暂时无法决定函数中的执行语句,可加一条 pass 语句使程序能够执行。

6-5 允许。在 Python 程序中,不仅允许函数 A 调用函数 B,函数 B 再调用函数 A,还允许一个函数自己调用自己,这称为递归。有关递归的概念及其使用将在后面“算法分析与设计”章节中介绍。

6-6

```
def line(n):
    for i in range(n):
        print('+',end='')
for i in range(1,9):
    line(i)
    print()
```

6-7

```
from math import sqrt
def sqrt_number(a):
    if a<0:
        return -1
    else:
        return sqrt(a)
x=float(input("please input number:"))
y=sqrt_number(x)
if y<0:
    print("无实数解!")
else:
    print(x,"的平方根是",y)
```

第 7 章

7-1 分别按占用的时间由小到大排列:

$10<O(\log_2 n)<O(n^{2/3})<O(18n)<O(6n^2)<O(2^n)<O(3^n)<O(n!)$

7-2 在最好和最坏情况下使用的比较次数分别是 $n-1$ 和 $2(n-1)$;而平均比较次数是 $3(n-1)/2$,因为在比较过程中,将有一半的几率出现 $A[i]>\text{maxval}$ 的情况。

7-3　折半查找平均情况下不成功检索的时间性能为判定树深度：$\log_2(n+1)$向上取整。

7-4　冒泡排序与选择排序在一般情况下哪一个效率更高些？

通过以下比较可知，在最坏情形下，选择排序在一般情况下效率更高些。

冒泡排序：

$$\text{KCN} = \sum_{i=1}^{n-1}(n-i) = \frac{1}{2}n(n-1)$$

$$\text{RMN} = 3\sum_{i=1}^{n-1}(n-i) = \frac{3}{2}n(n-1)$$

选择排序：

$$\text{KCN} = \sum_{i=0}^{n-2}(n-i-1) = \frac{n(n-1)}{2}$$

$$\text{RMN} = 3(n-1)$$

7-5　请分别用冒泡、选择、插入和快速排序法升序排列下面的实例，给出每一趟排序的结果。

(6,22,9,10,4,34,27,19,15,6)

请参照例 7-3-1、例 7-3-2、例 7-3-3、例 7-3-4。

7-6　在最坏情形下，算法的效率为$O(n^2)$。

对 n 个元素的序列进行排序所需的时间 $T(n)$的递推关系为

$$T(n) = T(n-1) + cn$$
$$T(n-1) = T(n-2) + c(n-1)$$
$$T(n-2) = T(n-3) + c(n-2)$$
$$\vdots$$
$$T(2) = T(1) + c(2)$$

将上等式都相加。

$$T(n) = T(1) + c(2+3+4+5+6+\cdots+n)$$

故时间性能为

$$T(n) = O(n^2)$$

7-7　分治法的基本思想是将一个规模为 n 的问题分解为与原问题相同的 k 个规模较小且互相独立的子问题。

7-8　一个直接或间接地调用自身的算法称为递归，它有两个条件，一个是要直接或间接地调用自身，另一个是必须有出口。

第 8 章

8-1

(1) C　(2) A　(3) B　(4) D　(5) D　(6) C　(7) C　(8) B　(9) A　(10) D

8-2

(1) 方法是面向对象中的术语，是用于描写对象动态特性(行为特性)的一组操作。对象方法是抽象对象的组成部分，是属于所定义类对象的，对象方法可以被类的实例对象调用。

函数是面向过程中的术语，是实现某种功能的一组代码。一般函数是公共的，不隶属于某类事物，可以被程序直接调用。

此外，在 Python 中，定义对象方法时，第一参数必须明确用 self 指明实例对象本身，而一般函数定义时无此要求。

（2）所谓继承，就是在现有类的基础上创建一个新类，所创建的新类从原来类那里获取已有特性，并可通过添加新特性对原来类进行扩展。被继承的类称为“基类”或“父类”，通过继承创建的新类称为“派生类”或“子类”。

继承是面向对象极为重要的特征，类的继承反映了事物之间的层次关系，体现了自然界中特殊与一般的关系。在软件开发过程中，继承保证了软件模块的可重用性和独立性，可缩短开发周期，提高软件开发效率，同时使软件易于扩展和维护。

（3）类变量是类所有对象所共有的，一个类的变量一经定义后，在其生存期间，对于任何实例对象，其属性值是相同的，若类的任一个实例对象改变了其值，则其他对象所获得的就是改变后的属性值；而对象（实例）变量则属实例对象自己拥有，某一个对象对其对象变量所做的改变，不影响其他对象。

类变量通常用于描述类所有对象的共同特征，它是为整个类服务的而非为某个对象服务的。

（4）Tkinter 创建 GUI 程序的基本步骤如下：

① 导入 tkinter 模块。

② 创建根窗口（顶层窗口）对象。

③ 在根窗口对象上创建所需的窗口“控件”对象。

④ 将窗口“控件”对象与对应事件处理程序代码相关联。

⑤ 进入窗口事件主循环。

（5）用户在图形界面上所做的鼠标或键盘等操作称为事件（当然还有系统事件等）。当某个事件发生时，应用程序启用相应的处理来响应事件。由于事件的发生顺序是无法预测的，所以事先将相应的处理代码与相关事件绑定，当事件触发时，就调用相应的代码进行处理，处理完毕，等待下一个事件。也即在事件驱动的应用程序中，程序不是按照预定的执行顺序运行的，而是在响应不同的事件时执行不同的代码。

附录 B　Python 习题选编

一、程序结构与算法部分

1. 编写一个 Python 程序,输入两个数,比较它们的大小并输出其中的较大者。

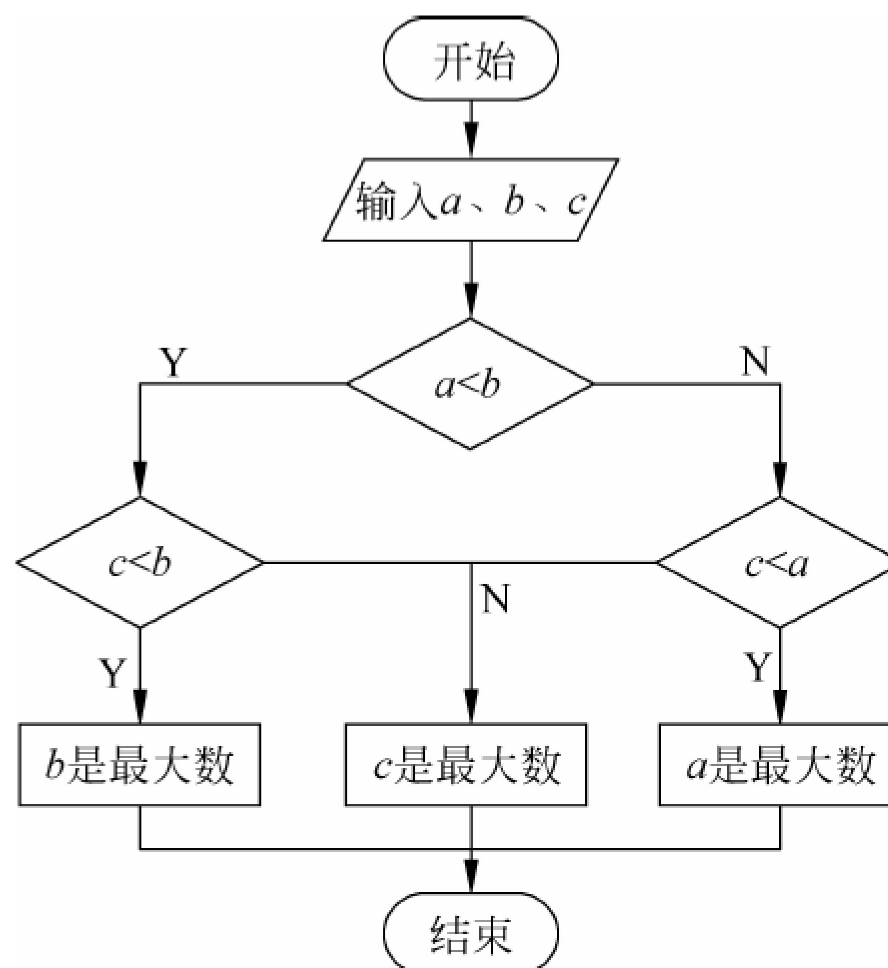
参考代码:

```
x = int(input("Please enter first integer: "))
y = int(input("Please enter second integer: "))
if (x == y):
    print("两数相同!")
elif (x > y):
    print("较大数为: ",x)
else:
    print("较大数为: ",y);
```

2. 写一个算法(流程图和 Python 程序):输入三个数,输出其最大者。

参考代码:

```
a,b,c = 3,4,5
if a <=  b:
    if c < b:
        print ("b 是最大的数")
    else:
        print ("c 是最大的数")
else:
    if c < a:
        print ("a 是最大的数")
    else:
        print ("c 是最大的数")
```



3. 使用 Python 编程,求 1～100 所有偶数的和。

参考代码:

```
sum = 0
for x in range(1,101):
   if x % 2 == 0:
       print(x)
       sum = sum + x
print("累加和是:",sum)
```

4. 用 Python 编写程序，输入一年份，判断该年份是否是闰年并输出结果。

注：凡符合下面两个条件之一的年份都是闰年。

（1）能被 4 整除但不能被 100 整除。

（2）能被 400 整除。

参考代码：

```
year = int(input("Please enter the year: "))
if ((year % 4 == 0 and year % 100!= 0) or (year % 400 == 0)):
    print(year,"is a leap year.")
else:
    print(year,"is not a leap year.")
```

5. 用 Python 编程，假设一年期定期利率为 3.25%，计算一下需要过多少年，一万元的一年定期存款连本带息能翻番？

参考代码：

```
cunkuan = 10000  # 本金 10000 元
years = 0
while cunkuan < 20000:
        years += 1
        cunkuan = cunkuan * (1 + 0.0325)
print(str(years) + "年以后，存款会翻番")
```

6. 从键盘接收一百分制成绩（0～100），要求输出其对应的成绩等级 A～E。其中，90 分以上为 A，80～89 分为 B，70～79 分为 C，60～69 分为 D，60 分以下为 E。

参考答案：

```
score = int(input('请输入成绩(0～100): '))
if score > 100:
    grade = "输入错误!"
elif score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
elif score >= 0:
    grade = 'E'
else:
    grade = "输入错误!"
print(grade)
```

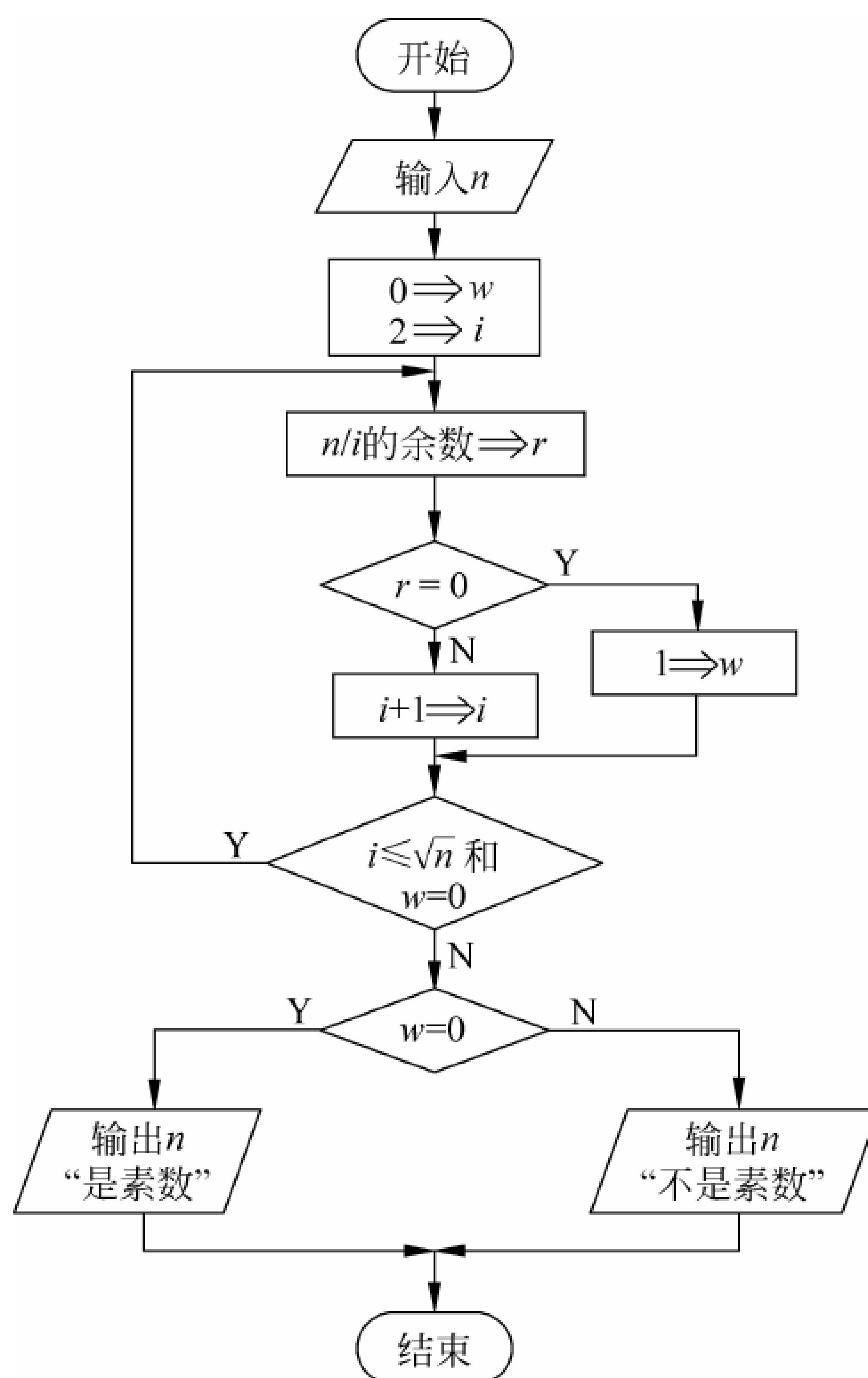
7. 猜数游戏。预设一个 0～9 的整数，让用户猜一猜并输入所猜的数，如果大于预设的数，则显示“太大”；如果小于预设的数，则显示“太小”，如此循环，直至猜中该数，显示“恭喜！你猜中了！”。

参考答案：

```
num = 7
while True:
    guess = int(input('请输入你猜的数(0～9): '))
    if guess == num:
        print("恭喜!你猜中了!")
        break;
    elif guess > num:
        print("太大")
    else:
        print("太小")
```

8. 输入一个数,判断这个数是否为素数,并输出判断结果。

所谓素数,是指除了 1 和该数本身之外,不能被其他任何整数整除的数。下图为参考流程图。



参考答案:

```
import math
n = int(input("请输入一个数:"))
x = int(math.sqrt(n))
i,w = 2,0
for i in range(2,x + 1):
    if n % i == 0:
        w = 1
if w == 1:
    print(n,"不是素数.")
```

```
else:
    print(n,"是素数.")
```

或

```
import math
n = int(input('请输入一个数: '))
i,w = 2,0
while i <= int(math.sqrt(n)) and w == 0:
    if n % i == 0:
        w = 1
        break
    else:
        i = i + 1
if w == 0:
    print(n,"是素数!")
else:
    print(n,"不是素数!")
```

或

```
import math
n = int(input('请输入一个数: '))
i = 2
while i <= int(math.sqrt(n)):
    if n % i == 0:
        print(n,"不是素数!")
        break
    else:
        i = i + 1
else:
    print(n,"是素数!")
```

9. 输入一个时间(小时:分钟:秒),输出该时间经过5分30秒后的时间。

参考答案:

```
hour,minute,second = input('请输入一个时间(h:m:s):').split(':')
hour = int(hour)
minute = int(minute)
second = int(second)
second += 30
if second >= 60:
    second = second - 60
    minute += 1
minute += 5
if minute >= 60:
    minute = minute - 60
    hour += 1
if hour == 24:
    hour = 0
print('%d:%d:%d' % (hour,minute,second))
```

10. 一个数如果恰好等于它的因子之和,这个数就称为"完数"。例如,6的因子为1、2、

3,而 6=1+2+3,因此 6 是完数。编程,找出 1000 之内的所有完数,并输出该完数及对应的因子(枚举法)。

参考答案:

```
m = 1000
for a in range(2,m + 1):
    s = a
    L1 = []
    for i in range(1,a):
        if a % i == 0:
            s -= i
            L1.append(i)
    if s == 0:
        print("完数: %d,因子包括: " % a,end = "")
        for j in range(1,len(L1)):
            print(" %d" % L1[j],end = ",")
        print("\n")
```

11. 编程,解决猴子吃桃问题。

猴子第一天摘下若干个桃子,当即吃了一半,还不过瘾,又多吃了一个。第二天早上又将剩下的桃子吃掉一半,又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上想吃时,只剩下一个桃子了。求第一天共摘多少个桃子(迭代法)。

参考答案:

```
day = 9
x = 1
while day > 0:
    x = (x + 1) * 2
    day -= 1
print("total = ",x)
```

二、输入输出与文件部分

1. 编写一个 Python 程序,输入两个数,输出两数之和。

参考代码:

```
x = int(input("Please enter first integer: "))
y = int(input("Please enter second integer: "))
print("The sum is:");
print(x + y);
```

2. 在当前目录下有一个文件名为 temp.txt 的文件,存放着上海从 2014 年 3 月 10 日(周一)到 3 月 16 日(周日)间一周的最高和最低气温(单位为℃)。其中,第一行为最高气温,第二行为最低气温。编程,找出这一周中第几天最热(按最高气温计算)。最高多少度?这一周中第几天最冷(按最低气温计算)?最冷多少度?

参考答案:

```
flname = "temp.txt"
f = open(flname)
ht = (f.readline()).strip()
```

```
L1 = list(ht.split(','))
lt = (f.readline()).strip()
L2 = list(lt.split(','))
f.close()
for i in range(len(L1)):
    L1[i] = int(L1[i])
    L2[i] = int(L2[i])
maxVal = L1[0]
maxDay = 0
minVal = L2[0]
minDay = 0
for i in range(1,len(L1)):
    if L1[i]> maxVal:
        maxVal = L1[i]
        maxDay = i
    if L2[i]< minVal:
        minVal = L2[i]
        minDay = i
print("这周第" + str(maxDay + 1) + "天最热,最高" + str(maxVal) + "摄氏度")
print("这周第" + str(minDay + 1) + "天最冷,最低" + str(minVal) + "摄氏度")
```

3. 在第2题的基础上,求出全周的平均气温(这一周各天平均温度的平均值,取整数)。假设在气象意义上,入春标准是连续5天日均气温超过10℃,根据这一周的气象数据是否能判断上海已经入春?

参考答案:

```
flname = "temp.txt"
f = open(flname)
ht = (f.readline()).strip()
L1 = list(ht.split(','))
lt = (f.readline()).strip()
L2 = list(lt.split(','))
f.close()
L3 = []
for i in range(len(L1)):
    L1[i] = int(L1[i])
    L2[i] = int(L2[i])
    L3.append(int((L1[i] + L2[i])/2))
sum = 0
k = 0
for i in range(len(L3)):
    sum = sum + L3[i]
    if L3[i]>= 10:
        k += 1
    else:
        k = 0
avg = int(sum/len(L3))
print("周平均气温为: ",avg)
if k >= 5:
    print("上海这周已入春.")
```

```
else:
    print("上海这周末入春.")
```

4. 当前目录下有一个文件名为 score1.txt 的文本文件,存放着某班学生的计算机课成绩,共有学号、平时成绩、期末成绩三列。请根据平时成绩占 40%,期末成绩占 60%的比例计算总评成绩(取整数),并分学号、总评成绩两列写入另一文件 score2.txt。同时在屏幕上输出学生总人数,按总评成绩计 90 以上、80~89、70~79、60~69、60 分以下各成绩档的人数和班级总平均分(取整数)。

参考答案:

```
f = open("score1.txt")
a = f.readline()
line = (f.readline()).strip()
f2 = open("score2.txt", 'w')
f2.write("学号平均成绩\n");
L2 = [0,0,0,0,0]
count = 0
sum = 0
while (len(line)!= 0):
    #print(line)
    L1 = line.split()
    f2.write(L1[0] + " ")
    f_score = int(int(L1[1]) * 0.4 + int(L1[2]) * 0.6)
    if 90 < f_score <= 100:
        L2[0] += 1
    elif f_score >= 80:
        L2[1] += 1
    elif f_score >= 70:
        L2[2] += 1
    elif f_score >= 60:
        L2[3] += 1
    else:
        L2[4] += 1
    count += 1
    sum += f_score
    f2.write(str(f_score) + "\n")
    line = (f.readline()).strip()
f.close()
f2.close()
avg_score = int(sum/count)
print("学生总人数为%d,按总评成绩计,90 以上%d 人、80~89 间%d 人、70~79 间%d 人、60~69
间%d 人、60 分以下%d 人.班级总平均分为%d 分." % (count, L2[0], L2[1], L2[2], L2[3], L2[4],
avg_score))
```

或

```
f = open("score1.txt")
a = f.readlines()
del a[0]
L3 = []
for line in a:
```

```
        line = line.strip()
        L1 = line.split()
        f_score = int(int(L1[1]) * 0.4 + int(L1[2]) * 0.6)
        L3.append([L1[0],f_score])
    f.close()
    c = [0,0,0,0,0]
    count = 0
    sum = 0
    f2 = open("score2.txt",'w')
    f2.write("学号   平均成绩\n");
    for L2 in L3:
        if 90 < L2[1]<= 100:
            c[0] += 1
        elif L2[1]>= 80:
            c[1] += 1
        elif L2[1]>= 70:
            c[2] += 1
        elif L2[1]>= 60:
            c[3] += 1
        else:
            c[4] += 1
        count += 1
        sum += L2[1]
        f2.write(L2[0] + " " + str(L2[1]) + "\n")
    f2.close()
    avg_score = int(sum/count)
    print("学生总人数为%d,按总评成绩计,90以上%d人、80～89间%d人、70～79间%d人、60～69间%d人、60分以下%d人.班级总平均分为%d分." % (count,c[0],c[1],c[2],c[3],c[4],avg_score))
```

5. 当前目录下有一个文本文件sample12.txt,其内容包含小写字母和大写字母。请将该文件复制到另一文件sample12_copy.txt,并将原文件中的小写字母全部转换为大写字母,其余格式均不变。

参考答案:

```
f = open("sample12.txt")
L1 = f.readlines()
f2 = open("sample12_copy.txt",'w')
for line in L1:
    f2.write(line.upper())
f.close()
f2.close()
```

6. 当前目录下有一个文件名为class_score.txt的文本文件,存放着某班学生的学号、数学课成绩(第二列)和语文课成绩(第三列)。请编程完成下列要求:

(1) 分别求这个班数学和语文的平均分(保留一位小数)并输出。

(2) 找出两门课都不及格(<60)的学生,输出他们的学号和各科成绩。

(3) 找出两门课的平均分在90分以上的学生,输出他们的学号和各科成绩。

建议用三个函数分别实现以上要求。

参考答案:

```
def output_avg(L):
    sum1,sum2 = 0,0
    for line in L:
        L1 = line.strip().split()
        sum1 += int(L1[1])
        sum2 += int(L1[2])
    count = len(L)
    avg1 = round(sum1/count,1)
    avg2 = round(sum2/count,1)
    print("这个班的数学平均分为: %4.1f,语文平均分为: %4.1f" % (avg1,avg2))

def output_notpass(L):
    print("两门课均不及格的学生学号及数学、语文成绩为:")
    for line in L:
        L1 = line.strip().split()
        if int(L1[1])<60 and int(L1[2])<60:
            print(line)

def output_good(L):
    print("两门课平均分在 90 分以上的学生学号及数学、语文成绩为:")
    for line in L:
        L1 = line.strip().split()
        f_score = round((int(L1[1]) + int(L1[2]))/2)
        if f_score >= 90:
            print(line)

f = open("class_score.txt")
L = f.readlines()
del L[0]
output_avg(L)
output_notpass(L)
output_good(L)
```

三、算法分析与设计部分

1. 编程,从键盘接收若干个整数(直接按 Enter 键表示结束),用冒泡法或选择法进行排序(从小到大),并将排序结果在屏幕上输出。同时估计算法的复杂度。

参考答案:

1) 选择排序

```
def bubble(List):
    num = len(List)
    for i in range(0,num - 1):
        for j in range(i + 1,num):
            if List[i]> List[j]:
                List[i],List[j] = List[j],List[i]
    return List

L1 = []
```

```
num_str = input('请输入一个需排序的整数：')
while len(num_str)!= 0:
    L1.append(int(num_str))
    num_str = input('请输入一个需排序的整数：')
print('排序后结果:', bubble(L1))
```

2）冒泡排序

```
def bubble(List):
    for i in range(0,len(List) - 1):
        for j in range(len(List) - 1,i, - 1):
            if List[j - 1]> List[j]:
                List[j - 1],List[j] = List[j],List[j - 1]
    return List
L1 = []
num_str = input('请输入一个需排序的整数：')
while len(num_str)!= 0:
    L1.append(int(num_str))
    num_str = input('请输入一个需排序的整数：')
print('排序后结果:', bubble(L1))
```

上述两个程序算法的时间复杂度均为 $O(n^2)$。

2. 从键盘接收一个正整数 n，输出对应斐波那契(Fibonacci)数列的前 n 项(计算数列中某项的值请用递归函数实现)。另外，请指出所用算法的复杂度。

有能力的同学还可进一步改进算法的效率。

参考答案：

```
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n - 1) + fib(n - 2)

n = int(input('n = '))
for i in range(n + 1):
    print(fib(i), end = " ")
```

算法时间复杂度为 $O(n \times 2^n)$。

可改进为：

```
def fib(n, List):
    a, b = 0, 1
    List.append(a)
    while b <= n:
        List.append(b)
        a, b = b, a + b
n = int(input('n = '))
L1 = []
fib(n, L1)
print(L1)
```

算法的时间复杂性为 $O(n)$。

3. 当前目录下有一个文件名为 score2.txt 的文本文件,存放着某班学生的计算机课成绩,共有学号、总评成绩两列。请查找最高分和最低分的学生,并在屏幕上显示其学号和成绩。另外,请指出所用算法的复杂度。

参考答案:

```
f = open("score2.txt")
a = f.readlines()
del a[0]
L2 = []
L3 = []
for line in a:
    line = line.strip()
    L1 = line.split()
    L2.append(L1[0])
    L3.append(L1[1])
f.close()
maxScore = L3[0]
maxIndex = 0
minScore = L3[0]
minIndex = 0
for i in range(1,len(L3)):
    if L3[i]>maxScore:
        maxScore = L3[i]
        maxIndex = i
    if L3[i]<minScore:
        minScore = L3[i]
        minIndex = i
print("最高分为: " + str(maxScore) + "分,该学生学号为: " + str(L2[maxIndex]))
print("最低分为: " + str(minScore) + "分,该学生学号为: " + str(L2[minIndex]))
```

时间复杂度为 $O(n)$。

四、数据结构部分

1. 编程,输入一行字符,分别统计出其中英文字母、空格、数字和其他字符的个数。

参考答案:

```
letters,space,digit,other = 0,0,0,0
s = input("请输入一行字符: ")
for i in range(len(s)):
    if (s[i]>= 'a' and s[i]<= 'z') or (s[i]>= 'A' and s[i]<= 'Z'):
        letters += 1
    elif s[i] == ' ':
        space += 1
    elif s[i]>= '0' and s[i]<= '9':
        digit += 1
    else:
        other += 1
print("字母数: %d\n空格数: %d\n数字数: %d\n其他字符数: %d\n" %(letters,space,digit,
other))
```

或

```
p = input('请输入一行字符:')
a,b,c,d = 0,0,0,0
for i in p:
    if((i <= 'Z' and i >= 'A') or (i <= 'z' and i >= 'a')):
        a += 1
    elif (i == ' '):
        b += 1
    elif(i >= '0' and i <= '9'):
        c += 1
    else:
        d += 1
print('英文字母的个数为: ' + str(a))
print('空格的个数为: ' + str(b))
print('数字的个数为: ' + str(c))
print('其他字符的个数为: ' + str(d))
```

或

```
letter,space,digit,other = 0,0,0,0
s = input('input a string:')
for c in s:
    if (c <= 'Z' and c >= 'A')or(c <= 'z' and c >= 'a'):
        letter += 1
    elif c.isspace():
        space += 1
    elif c.isdigit():
        digit += 1
    else:
        other += 1
print("字母数: %d\n空格数: %d\n数字数: %d\n其他字符数: %d\n" % (letter,space,digit,
other))
```

2. 小王希望用电脑记录他每天掌握的英文单词。请设计程序和相应的数据结构,使小王能记录新学的英文单词和其中文翻译,并能很方便地根据英文来查找中文。

【参考:数据结构建议用字典。字典添加:dic[key]=value;判断 key 是否在字典中:if key in dic。】

参考答案:

```
def add_dic(dic):
    while True:
        word = input("请输入英文单词(直接按 Enter 键结束): ")
        if len(word) == 0:
            break;
        meaning = input("请输入中文翻译: ")
        dic[word] = meaning
        print("该单词已添加到字典库.")
    return
def search_dic(dic):
    while True:
```

```
        word = input("请输入要查询的英文单词(直接按 Enter 键结束): ")
        if len(word) == 0:
            break;
        if word in dic:
            print("%s 的中文翻译是 %s" % (word,dic[word]))
        else:
            print("字典库中未找到这个单词")
    return

worddic = dict()
while True:
    print("请选择功能: \n1:输入\n2: 查找\n3: 退出")
    c = input()
    if c == "1":
        add_dic(worddic)
    elif c == "2":
        search_dic(worddic)
    elif c == "3":
        break
    else:
        print("输入有误!")
```

五、异常处理部分

1. 以下是两数相加的程序:

```
x = int(input("x = "))
y = int(input("y = "))
print("x + y = ",x + y);
```

该程序要求接收两个整数,并输出相加结果。但如果输入的不是整数(如字母、浮点数等),程序就会终止执行并输出异常信息。请对程序进行修改,要求输入非整数时,给出“输入内容必须为整数!”的提示,并提示用户重新输入,直至输入正确。

参考答案:

```
while True:
    try:
        x = int(input("x = "))
    except ValueError:
        print("输入内容必须为整数!")
    else:
        break
while True:
    try:
        y = int(input("y = "))
    except ValueError:
        print("输入内容必须为整数!")
    else:
        break
print("x + y = ",x + y)
```

2. 编程,请输入一个文件路径名或文件名,查看该文件是否存在,如存在,则打开文件

并在屏幕上输出该文件内容；如不存在，则显示“输入的文件未找到！”并要求重新输入；如文件存在但在读文件过程中发生异常，则显示“文件无法正常读出！”并要求重新输入。

【提示：请使用异常处理。“文件未找到”对应的异常名为 FileNotFoundError，其他异常直接用 except 匹配。】

参考答案：

```
while True:
    try:
        filename = input('请输入文件路径名或文件名：')
        f = open(filename.strip())
        print(f.read())
    except FileNotFoundError:
        print("输入的文件未找到!")
    except:
        print("文件无法正常读出!")
    else:
        break
f.close()
```

六、函数部分

1. 写一判素数的函数，在主函数中输入一个整数，调用该函数进行判断并输出结果。

参考答案：

```
def shushu(n):
    import math
    i,w = 2,0
    if n <= 1:
        w = 1
    while i <= int(math.sqrt(n)) and w == 0:
        if n % i == 0:
            w = 1
            break
        else:
            i = i + 1
    return w

n = int(input('n = '))
if shushu(n) == 0:
    print(n,"是素数!")
else:
    print(n,"不是素数!")
```

2. 当前目录下有一个文件名为 score3.txt 的文本文件，存放着某班学生的学号和其两门专业课的成绩，分别用函数实现以下功能：

(1) 定义函数 function1，计算每个学生的平均分(取整数)，并将所有学生的学号和平均分在屏幕上输出(函数参数为要读取文件的文件名)。

```
def function1(flname):
    #函数代码
```

```
function1("c:\\test\\score3.txt")
```

参考答案:

```
def function1(flname):
    f = open(flname)
    a = f.readlines()
    del a[0]
    L3 = []
    for line in a:
        line = line.strip()
        L1 = line.split()
        avg_score = int((int(L1[1]) + int(L1[2]))/2)
        L3.append([L1[0],avg_score])
    f.close()
    print("学号  平均分")
    for L2 in L3:
        print(L2[0] + " " + str(L2[1]))
```

(2) 定义函数 calAvg(),计算某一门课程的平均分(函数参数为某门课成绩对应的列表名,返回值为该门课的平均分)。

```
def calAvg(L):
    #函数代码

f = open("c:\\test\\score3.txt")
a = f.readlines()
del a[0]
L2 = []
L3 = []
for line in a:
    line = line.strip()
    L1 = line.split()
    L2.append(int(L1[1]))
    L3.append(int(L1[2]))
f.close()
print("专业课 1 的总平均分为",calAvg(L2))
print("专业课 2 的总平均分为",calAvg(L3))
```

参考答案:

```
def calAvg(L):
    sum,count = 0,0
    for score in L:
        sum += score
        count += 1
    avg_score = int(sum/count)
    return avg_score
```

3. 用函数或函数的递归实现求 $n!$ 的算法(主程序已给出),同时估计程序的复杂度。

```
def fact(n):
    //函数代码
n = int(input("Calculate n! Enter n = "))
```

```
print(n,'!= ',fact(n))
```

参考答案：

```
def fact(n):
    value = 1
    for count in range(1,n+1):
        value *= count
    return value
```

或

```
def fact(n):
    if n == 1:
        value = 1
    else:
        value = n * fact(n-1)
    return value
```

程序复杂度为 $O(n)$。

4. 分别编写求两个整数的最大公约数的函数 hcf 和求最小公倍数的函数 lcd。主函数已给出，其从键盘接收两个整数，调用这两个函数后输出结果。

【提示：求最大公约数可用辗转相除法。即将大数作为被除数，小数作为除数，若两者余数不为 0，则将小数作为被除数，余数作为除数……直到余数为 0。求最小公倍数则用两数的积除以最大公约数即可。】

参考答案：

```
def hcf(u,v):
    if v > u:
        u,v = v,u
    r = u % v
    while r!= 0:
        u = v
        v = r
        r = u % v
    return v

def lcd(u,v,h):
    return u * v/h

u = int(input("请输入第一个整数："))
v = int(input("请输入第二个整数："))
h = hcf(u,v)
print("%d 和 %d 的最大公约数为 %d: " % (u,v,h))
l = lcd(u,v,h)
print("%d 和 %d 的最小公倍数为 %d: " % (u,v,l))
```

5. 编程，统计列表中各数据的方差和标准差。主函数已给出，请编写计算方差的函数 var。

【提示：方差的计算公式为 $\sum X_i^2/n-(\sum X_i/n)^2$，其中，$n$ 为列表中的元素个数，X_i 为

列表中的第 i 项。标准差则为方差的算术平方根。】

参考答案：

```
import math
def var(L1):
    s,psum = 0,0
    for i in range(len(L1)):
        v = L1[i]
        s += v
        psum += v * v
    s = s/len(L1)
    mse = psum/len(L1) - s * s
    return mse

L1 = [5,3,7,8,14,9,12,6]
dx = var(L1)
print('方差为: %.2f' % dx)
mse = math.sqrt(dx)
print('标准差为: %.2f' % mse)
```

6. 主程序中已有一个排好序的列表，请编写函数 insertList，将从键盘接收的整数按原来从小到大的排序规律插入该列表中。

```
def insertList(L1,x):
    #函数代码

L1 = [1,4,6,9,13,16,28,40,100]
x = int(input('请输入一个要插入的整数: '))
insertList(L1,x)
print(L1)
```

参考答案：

```
def insertList(L1,x):
    if x > L1[len(L1) - 1]:
        L1.append(x)
        return
    for i in range(0,len(L1)):
        if x < L1[i]:
            L1.insert(i,x)
            break
    return

L1 = [1,4,6,9,13,16,28,40,100]
x = int(input('请输入一个要插入的整数: '))
insertList(L1,x)
print(L1)
```

参 考 文 献

[1] 陆朝俊. 程序设计思想与方法：问题求解中的计算思维[M]. 北京：高等教育出版社，2013.
[2] Allen B Downey. Python for Software Design：How to Think Like a Computer Scientist. London：Cambridge University Press，2009.
[3] [美]Wesley J Chun. Python 核心编程(第二版)[M]. 宋吉广，译. 北京：人民邮电出版社，2008.
[4] [美] Grady Booch，等. 面向对象分析与设计(第三版)[M]. 王海鹏，等，译. 北京：人民邮电出版社，2009.
[5] 丁任霜，等. 面向对象设计与应用[M]. 北京：北京大学出版社，2011.
[6] 赵英良. 大学计算机基础(第 4 版). 西安：西安交通大学出版社，2011.
[7] [美]H M Deitel，P J Deitel，J P Liperi，B A Wiedermann. Python 编程金典. 周靖，译. 北京：清华大学出版社，2003.
[8] [美]Mark Summerfield. Python 3 程序开发指南. 王弘博，孙传庆，译. 北京：人民邮电出版社，2011.
[9] 许少华. 算法分析与设计. 哈尔滨：哈尔滨工业大学出版社，2011.
[10] 汪小林，罗英伟，李文新. 计算概论——程序设计阅读题解. 北京：清华大学出版社，2011.